TUGAS AKHIR – KI1502

BUSINESS PROCESS INTELLIGENCE BERDASARKAN GRAPH MODEL (STUDI KASUS: ENTERPRISE RESOURCE PLANNING RETAIL)

ANDY YOHANES HADIWIJAYA
NRP 5114 100 050

Dosen Pembimbing I
Prof. Drs. Ec. Ir. Riyanarto Sarno, M.Sc., Ph.D.

Dosen Pembimbing II
Dwi Sunaryono, S.Kom., M.Kom.

DEPARTEMEN INFORMATIKA
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember
Surabaya 2018

[Halaman ini sengaja dikosongkan]

TUGAS AKHIR – KI1502

BUSINESS PROCESS INTELLIGENCE BERDASARKAN GRAPH MODEL (STUDI KASUS: ENTERPRISE RESOURCE PLANNING RETAIL)

ANDY YOHANES HADIWIJAYA
NRP 5114 100 050

Dosen Pembimbing I
Prof. Drs. Ec. Ir. Riyanarto Sarno, M.Sc., Ph.D.

Dosen Pembimbing II
Dwi Sunaryono, S.Kom.,M.Kom.

DEPARTEMEN INFORMATIKA
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember
Surabaya 2018

[Halaman ini sengaja dikosongkan]

FINAL PROJECT – KI1502

BUSINESS PROCESS INTELLIGENCE BASED ON GRAPH MODEL (CASE STUDY: ENTERPRISE RESOURCE PLANNING RETAIL)

ANDY YOHANES HADIWIJAYA
NRP 5114 100 050

Supervisor I
Prof. Drs. Ec. Ir. Riyanarto Sarno, M.Sc., Ph.D.

Supervisor II
Dwi Sunaryono, S.Kom.,M.Kom.

DEPARTMENT OF INFORMATICS
Faculty of Information and Communication Technology
Institut Teknologi Sepuluh Nopember
Surabaya 2018

[Halaman ini sengaja dikosongkan]

LEMBAR PENGESAHAN

BUSINESS PROCESS INTELLIGENCE BERDASARKAN GRAPH MODEL (STUDI KASUS: ENTERPRISE RESOURCE PLANNING RETAIL)

TUGAS AKHIR

Diajukan Untuk Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Rumpun Mata Kuliah Manajemen Informasi
Sistem Studi S-1 Departemen Informatika
Fakultas Teknologi Informasi dan Komunikasi
Institut Teknologi Sepuluh Nopember

Oleh
ANDY YOHANES HADIWIJAYA
NRP. 5114 100 050

Disetujui oleh Dosen Pembimbing Tugas Akhir:

Prof. Drs. Ec. Ir. RIYANARTO SARNO, M.Sc., Ph.D.
NIP: 19590803 198601 1 001 ..... (pembimbing 1)

DWI SUNARYONO, S.Kom., M.Kom.
NIP: 198608232015041004 ..... (pembimbing 2)

SURABAYA
JANUARI, 2018

[Halaman ini sengaja dikosongkan]

BUSINESS PROCESS INTELLIGENCE BERDASARKAN GRAPH MODEL (STUDI KASUS: ENTERPRISE RESOURCE PLANNING RETAIL)

Nama : Andy Yohanes Hadiwijaya
NRP : 5114100050
Departemen : Informatika – FTIK
Dosen Pembimbing I: Prof. Drs. Ec. Ir. Riyanarto Sarno, M.Sc., Ph.D.
Dosen Pembimbing II : Dwi Sunaryono, S.Kom., M.Kom.

Abstrak

Event Log merupakan salah satu aset penting di perusahaan. Tugas akhir ini berusaha untuk memodelkan model proses bisnis di *event log* sehingga bisa menganalisa proses model agar bisa membawa berbagai manfaat bagi perusahaan seperti mengurangi biaya perusahaan, membuat keuntungan meningkat, dan banyak lagi. Salah satu masalah dalam pemodelan *event log* secara otomatis adalah penambahan *non-free choice*. Tugas akhir ini bertujuan untuk mengembangkan sebuah sistem yang dapat memodelkan proses bisnis dari *event log* yang berisi *non-free choice* dengan menggunakan *graph database* di Neo4j. Pertama-tama, penelitian ini menghasilkan *event log* ke dalam *link list* berupa grafik di Neo4j. Selanjutnya, penelitian ini mengusulkan sebuah metode untuk meningkatkan *link list* dari *event log* menjadi *graph model* yang mengandung *non-free choice*. Hasil *graph model* yang diperoleh dengan menggunakan Neo4j dibandingkan dengan hasil yang diperoleh dengan menggunakan LTL yang diturunkan dari aturan dalam model deklaratif. Makalah ini dapat membuktikan bahwa metode yang diusulkan dengan menggunakan *graph database* adalah metode yang tepat karena hasil grafik memiliki hasil yang lebih baik daripada *Heuristic Miner*. Hasil Neo4j bisa langsung menghasilkan relasi yang benar meski *event log* mengandung *non-free choice* dimana tidak bisa dilakukan pada *Heuristic Miner*.

Kata kunci: *Control Flow Pattern, Non-free-choice, Neo4J, Graph Database*

[Halaman ini sengaja dikosongkan]

BUSINESS PROCESS INTELLIGENCE BERDASARKAN GRAPH MODEL (STUDI KASUS: ENTERPRISE RESOURCE PLANNING RETAIL)

Student Name	**: Andy Yohanes Hadiwijaya**
NRP	**: 5114100050**
Major	**: Informatics – FTIK**
Supervisor I	**: Prof. Drs. Ec. Ir. Riyanarto Sarno, M.Sc., Ph.D.**
Supervisor II	**: Dwi Sunaryono, S.Kom., M.Kom.**

Abstract

The event log is one of the important assets in the company. This research strives to model the business process model in event log so that can analyze the model process in order to bring various benefits to companies such as reducing company costs, increasing profits, and more. One of the problem in modeling event log automatically is the addition of non-free choice. This research aims to develop a system that can model a business process of an event log that contains non-free choice using a graph database in Neo4j. First, this research model the event log into a link list in the form of a graph in Neo4j. Next, this research proposes a method to enhance the link list of event log to be a graph model that contains the non-free-choice. The results of the obtained graph model using Neo4j are compared with the results that are obtained using LTL that is derived from rules in a declarative model. This research can prove that the proposed method using graph database is a right method because the graph result has a better result than a Heuristic Miner. The result of Neo4j can directly generate relation which is true even though event log containing non-free choice where can't be done in the Heuristic Miner.

Keywords: *Control Flow Pattern, Non-Free Choice, Neo4J, Graph Database*

[Halaman ini sengaja dikosongkan]

KATA PENGANTAR

Segala puji syukur penulis kepada Tuhan Yesus Kristus atas penyertaan-Nya, sehingga tugas akhir berjudul "B*usiness Process Intelligence* Berdasarkan *Graph Model* (Studi Kasus: *Enterprise Resource Planning Retail*)" ini dapat selesai sesuai dengan waktu yang telah ditentukan.

Pengerjaan tugas akhir ini menjadi sebuah sarana untuk penulis memperdalam ilmu yang telah didapatkan di Institut Teknologi Sepuluh Nopember Surabaya, khususnya dalam disiplin ilmu Teknik Informatika. Terselesaikannya buku tugas akhir ini tidak terlepas dari bantuan dan dukungan semua pihak. Pada kesempatan kali ini penulis ingin mengucapkan terima kasih kepada:

1. Mama, papa dan keluarga yang selalu memberikan dukungan berupa doa selama proses pengerjaan Tugas Akhir.
2. Bapak Riyanarto Sarno dan Bapak Dwi selaku dosen pembimbing yang telah bersedia meluangkan waktu selama proses pengerjaan Tugas Akhir.
3. Bapak dan Ibu dosen Jurusan Teknik Informatika ITS yang banyak memberikan ilmu dan bimbingan bagi penulis.
4. Seluruh staf dan karyawan FTIf ITS yang banyak memberikan kelancaran administrasi akademik kepada penulis.
5. Teman-teman TC Angkatan 2014 yang selalu mendukung selama proses pengerjaan tugas akhir.
6. Kelly Rossa, Dewi Rahmawati dan teman-teman RMP yang membantu penulis dalam pengerjaan tugas akhir.
7. Serta semua pihak yang turut membantu penulis dalam menyelesaikan tugas akhir ini.

Penulis menyadari bahwa tugas akhir ini masih memiliki banyak sekali kekurangan. Dengan kerendahan hati, penulis mengharapkan kritik dan saran dari pembaca untuk perbaikan ke depan.

Surabaya, Januari 2018

[*Halaman ini sengaja dikosongkan]*

DAFTAR ISI

[Halaman ini sengaja dikosongkan]

DAFTAR GAMBAR

[Halaman ini sengaja dikosongkan]

DAFTAR TABEL

[Halaman ini sengaja dikosongkan]

DAFTAR KODE SUMBER

[Halaman ini sengaja dikosongkan]

DAFTAR SINGKATAN

LTL	:	*Linear Temporal Logic*.
CSM	:	*Comma Separated Value*.
ProM	:	*Process Mining*.

[Halaman ini sengaja dikosongkan]

BAB I
PENDAHULUAN

Pada bab ini akan dipaparkan mengenai garis besar Tugas Akhir yang meliputi latar belakang, tujuan, rumusan dan batasan permasalahan, metodologi pengerjaan Tugas Akhir, dan sistematika penulisan Tugas Akhir.

1.1 Latar Belakang

Event log adalah penyimpanan informasi proses dalam sebuah sistem [1]. Informasi dasar yang disimpan oleh *event log* terdiri dari nama kegiatan, sumber daya yang melakukan aktivitas, dan waktu kegiatan. Potongan informasi tersebut digunakan dalam analisis proses dan penemuan pemecahan masalah [2]. Singkatnya, *event log* adalah bahan untuk menganalisa proses yang dijalankan dan menemukan masalah dalam proses. Untuk mengolah *event log* tersebut diperlukan *Business Process Intelligence*.

Business Process Intelligence adalah sebuah kumpulan metodologi, proses, arsitektur, dan teknologi yang mengubah data asli menjadi informasi yang berarti dan bermanfaat. Hal ini memungkinkan pengguna bisnis membuat keputusan bisnis dengan data *real-time* yang dapat menempatkan perusahaan di depan pesaingnya.

Tapi, kita punya banyak masalah untuk memodelkan proses bisnis dari *event log* seperti pemodelan *event log* yang mengandung *non-free choice*. Kita tidak bisa memodelkan *non-free choice* karena *non-free choice* tidak tercatat dalam *event log*. Selain masalah di atas, kita tidak bisa memodelkan *event log* ke model proses bisnis secara manual karena sangat merugikan di bidang waktu dan usaha.

Dari masalah itu maka kita membutuhkan sebuah sistem yang bisa memodelkan secara otomatis dan bisa menangani *event log* yang mengandung *non-free choice*. Oleh karena itu, makalah ini bertujuan untuk mengembangkan sebuah sistem yang dapat

memodelkan model proses bisnis dari *event log* yang mengandung *non-free choice* dengan menggunakan *graph database* di Neo4J.

1.2 Rumusan Permasalahan

Rumusan masalah yang diangkat dalam Tugas Akhir ini dapat dipaparkan sebagai berikut:

1. Bagaimana cara mentransform data dari *database* menjadi *event log*?
2. Bagaimana cara membentuk rule LTL (*Linear Temporal Logic*) berdasarkan *event log* yang mengandung *non-free choice* dengan menggunakan *Declarative Miner*?
3. Bagaimana cara membentuk *graph model* dari rule LTL (*Linear Temporal Logic*)?
4. Bagaimana cara membentuk *graph model* dari *event log* yang mengandung *non-free choice* di Neo4J?
5. Bagaimana cara membandingkan *graph model* yang berasal dari rule LTL dengan *graph model* dari *event log* ?

1.3 Batasan Permasalahan

Permasalahan yang dibahas dalam Tugas Akhir ini memiliki beberapa batasan, di antaranya sebagai berikut:

1. lmplementasi *Business Process Intelligence* dengan *database* Neo4j.
2. Data masukan berupa *event log* dalam bentuk Excel yang diproses menggunakan *Declarative Miner* pada Java sehingga membentuk suatu model.
3. Data masukan berupa *event log* dalam bentuk Excel yang diproses menggunakan *Cypher* pada Neo4J sehingga membentuk suatu model.
4. Data masukan berupa *event log* dalam bentuk Excel yang diproses menggunakan *Heuristic Miner* pada ProM sehingga membentuk suatu model.
5. Perbandingan dari model *Declarative Miner*, *Cypher* dan *Heuristic Miner* untuk mendapatkan hasil yang valid pada *Cypher*.

1.4 Tujuan

Tujuan dari Tugas Akhir ini adalah dapat membentuk *graph model* yang dapat menangani *non-free choice* dari *event log*, LTL, dan *Heuristic Miner* yang dapat dibandingkan untuk memperoleh hasil terbaik dari ketiga metode.

1.5 Manfaat

Penelitian ini diharapkan dapat membentuk *graph model* yang berasal dari kumpulan *control-flow patterns* yang dibentuk dari *Declarative Model* dan *event log*. Namun *graph model* yang dibentuk telah menangani *non-free choice*, sehingga data proses bisnis lebih mudah untuk diolah.

1.6 Metodologi

Langkah-langkah yang ditempuh dalam pengerjaan Tugas Akhir ini yaitu sebagai berikut:

1. **Studi literatur**

   Dalam pembuatan Tugas Akhir ini telah dipelajari tentang hal-hal yang dibutuhkan sebagai ilmu penunjang dalam penyelesaiannya. Ilmu penunjang utama pada Tugas Akhir ini adalah permodelan menggunakan Neo4j dengan *cypher* dan *Declarative Miner*. Selain itu, terdapat literatur lain yang menunjang proses penyelesaian Tugas Akhir ini.

2. **Pemodifikasian Metode**

   Pada tahap ini penulis menjabarkan cara pemecahan masalah yang terdapat dalam rumusan masalah.

3. **Analisis dan Perancangan Sistem**

   Aktor yang menjadi pelaku adalah pengguna perangkat lunak yang dibangun oleh penulis. Kemudian

beberapa kebutuhan fungsional dari sistem ini adalah sebagai berikut:

a. Preprocessing data yang akan digunakan sebagai *input*.
b. Menemukan *rule* LTL dari *event log* dengan menggunakan *Declarative Miner*.
c. Memodelkan *rule* LTL yang didapatkan dari *Declarative Miner* ke bentuk *graph* menggunakan Neo4j.
d. Memodelkan relasi *graph* dari *event log* yang mengandung *non-free choice* dengan *cypher.*
e. Menemukan *rule* LTL dari *event log* dengan menggunakan *Heuristic Miner*.
f. Membandingkan hasil dari *Declarative Miner*, *cypher* dan *Heuristic Miner*.

4. Implementasi

Pada tahap ini dilakukan pembuatan perangkat lunak yang menghasilkan dua *graph* yang bisa untuk menangani *non-free choice* dengan menggunakan Neo4j yang terbentuk dari hasil *Declarative Miner* dan *input* langsung dari *event log*, dan *Heuristic Miner* kemudian membandingkan ketiga hasil tersebut.

5. Pengujian dan evaluasi

Pada tahap ini dilakukan pengujian terhadap elemen perangkat lunak dengan menggunakan data uji yang telah dipersiapkan. Pengujian dan evaluasi perangkat lunak dilakukan untuk mengevaluasi jalannyaperangkat lunak, mengevaluasi fitur utama, mengevaluasi fitur-fitur tambahan, mencari kesalahan yang timbul pada saat perangkat lunak aktif, dan mengadakan perbaikan jika ada kekurangan. Tahapan-tahapan dari pengujian adalah sebagai berikut:

a. *Graph model* yang dibentuk dari *Linear Temporal Logic* maupun dari *event log* harus sudah menangani masalah *non-free choice*.

b. *Graph model* yang dibentuk dari *Linear Temporal Logic* maupun dari *event log* harus sama.

6. Penyusunan buku Tugas Akhir

Pada tahap ini dilakukan dokumentasi dan pelaporan dari seluruh konsep, dasar teori, implementasi, proses yang telah dilakukan, dan hasil-hasil yang telah didapatkan selama pengerjaan Tugas Akhir.

1.7 Sistematika Penulisan

Buku Tugas Akhir ini bertujuan untuk mendapatkan gambaran dari pengerjaan Tugas Akhir ini. Selain itu, diharapkan dapat berguna untuk pembaca yang tertarik untuk melakukan pengembangan lebih lanjut. Secara garis besar, buku Tugas Akhir terdiri atas beberapa bagian seperti berikut ini.

Bab I Pendahuluan

Bab ini berisi latar belakang masalah, tujuan dan manfaat pembuatan Tugas Akhir, permasalahan, batasan masalah, metodologi yang digunakan, dan sistematika penyusunan Tugas Akhir.

Bab II Dasar Teori

Bab ini membahas beberapa teori penunjang yang berhubungan dengan pokok pembahasan dan yang menjadi dasar dari pembuatan Tugas Akhir ini.

Bab III Metode Pemecahan Masalah

Bab ini membahas cara penulis memecahkan masalah yang ada. Penjelasan tentang algoritma yang dikembangkan penulis dan langkah-langkahnya sehingga dapat memecahkan masalah yang ada.

Bab IV Analisis dan Perancangan Sistem

Bab ini membahas mengenai perancangan perangkat lunak. Perancangan perangkat lunak

meliputi perancangan alur, proses dan perancangan antarmuka pada perangkat lunak.

Bab V Implementasi

Bab ini berisi implementasi dari perancangan perangkat lunak perangkat lunakdan implementasi fitur-fitur penunjang perangkat lunak.

Bab VI Pengujian dan Evaluasi

Bab ini membahas pengujian dengan metode pengujian subjektif untuk mengetahui penilaian aspek kegunaan (*usability*)dari perangkat lunak dan pengujian fungsionalitas yang dibuat dengan memperhatikan keluaran yang dihasilkan serta evaluasi terhadap fitur-fitur perangkat lunak.

Bab VII Kesimpulan

Bab ini berisi kesimpulan dari hasil pengujian yang dilakukan. Bab ini membahas saran-saran untuk pengembangan sistem lebih lanjut.

Daftar Pustaka

Merupakan daftar referensi yang digunakan untuk mengembangkan Tugas Akhir.

Lampiran

Merupakan lampiran yang digunakan untuk menjabarkan hasil pengujian algoritma.

BAB II
DASAR TEORI

Pada bab ini akan dibahas mengenai teori-teori yang menjadi dasar dari pembuatan Tugas Akhir.

2.1 *Business Process Intelligence*

Business Process Intelligence adalah metode yang sedang berkembang, yang dengan cepat dapat menarik perhatian karena meningkatnya tekanan yang dihadapi perusahaan untuk meningkatkan efisiensi proses bisnis sebuah perusahaan dan dengan cepat bereaksi terhadap perubahan pasar agar dapat bersaing di era internet yang sangat dinamis sekarang ini. BPI menyediakan beberapa fitur yang menawarkan berbagai tingkat otomatisasi untuk pengelolaan kualitas proses, diantaranya:

- Analisis

  BPI memungkinkan pengguna untuk menganalisis eksekusi proses yang selesai baik dari perspektif bisnis maupun informasi dan teknologi. Analis teknologi dan informasi akan tertarik untuk melihat informasi tingkat rendah dan terperinci seperti waktu eksekusi rata-rata per node atau panjang antrian pekerjaan, sedangkan pengguna bisnis akan tertarik pada informasi tingkat tinggi, seperti jumlah eksekusi proses 'sukses'. BPI juga menawarkan fitur untuk membantu analis dalam mengidentifikasi penyebab perilaku eksekusi proses yang diminati. Kemampuan analis BPI juga dapat diterapkan untuk menganalisis perancangan model proses dan khususnya untuk mengidentifikasi teknik untuk memperbaiki definisi proses yang ada.
- Prediksi

  BPI dapat menurunkan model prediksi dan menerapkan model semacam itu pada proses yang berjalan, untuk mengidentifikasi secara khusus kemungkinan pengecualian atau perilaku yang tidak diinginkan. Seperti dalam kasus analisis, prediksi dapat

dilakukan di tingkat teknologi dan informasi maupun di tingkat bisnis.

- Monitoring
  BPI dapat memantau dan menganalisis contoh proses yang berjalan dan menginformasikan pengguna tentang situasi yang tidak biasa atau tidak diinginkan. Pengguna dapat melihat status kesehatan sistem, proses, layanan dan sumber daya. Selain itu mereka dapat menentukan situasi kritis, sehingga BPI dapat memberi tahu mengenai media pilihan jika terjadi situasi kritis.
- Optimisasi
  BPI dapat mengidentifikasi bidang-bidang perbaikan dalam definisi proses bisnis dan dalam penugasan sumber daya dan layanan untuk kegiatan kerja.

2.2 *Process Discovery*

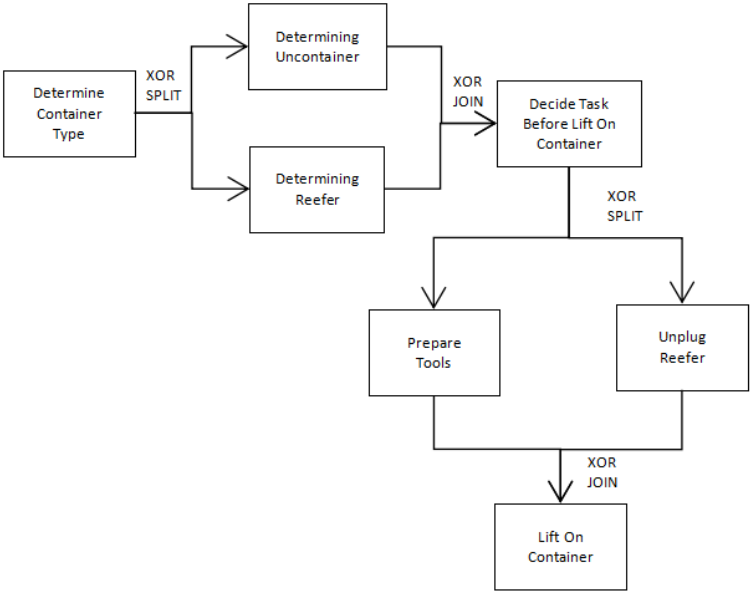
Process mining adalah suatu ilmu yang menggabungkan antara pembelajaran mesin serta *data mining* pada satu sisi dan memodelkan serta menganalisa proses pada sisi yang lain. Ide dasar dari *process mining* adalah menemukan, memantau serta meningkatkan proses sebenarnya dengan mengambil pengetahuan dari *event log* yang ada pada suatu sistem [2].

Process mining dibagi menjadi tiga bagian, salah satunya adalah *process discovery*. *Process discovery* adalah suatu teknik yang memanfaatkan *event log* dalam penentuan proses model tanpa menggunakan informasi-informasi lainnya. *Process discovery* yang ada diantaranya algoritma *Alpha++* [2] serta *Heuristic Miner* [2].

Proses model yang benar adalah proses model yang mampu menampilkan konkurensi serta *control-flow pattern* [1]. Konkurensi berarti aktivitas pada proses model tidak ditampilkan secara redundan (tidak ada aktivitas yang sama ditampilkan pada proses model) [2]. *Control-flow pattern* adalah penanda relasi antar aktivitas. Terdapat empat relasi antar aktivitas yaitu relasi *sequence*, relasi XOR, relasi AND and relasi OR [4].

Relasi *sequence* adalah relasi yang terjadi untuk menghubungkan satu aktivitas dengan satu aktivitas lainnya. Relasi

XOR adalah relasi yang terjadi yang hanya memperbolehkan salah satu aktivitas yang terjadi pada suatu proses. Relasi OR adalah relasi yang terjadi apabila beberapa aktivitas yang terjadi pada suatu proses. Sedangkan relasi AND adalah relasi yang terjadi dimana semua aktivitas harus dijalankan semuannya. Apabila aktivitas pilihan pada relasi XOR, OR dan AND adalah aktivitas sebelum dari aktivitas lainnya, maka *control-flow pattern* dari relasi tersebut adalah penambahan kata "Split" pada relasi tersebut [4]. Sedangkan, apabila aktivitas pilihan pada relasi XOR, OR dan AND adalah aktivitas sebelum dari aktivitas lainnya, maka *control-flow pattern* dari relasi tersebut adalah penambahan kata "Join" pada relasi tersebut [4]. Gambar 2.1 merupakan contoh penggunaan relasi pada proses model.



Gambar 2.1 Contoh Model Proses

2.3 *Streaming Event Log*

Event log merupakan catatan kejadiandari suatu proses bisnis yang berjalan [5]. Sebuah kejadian menunjukkan aktivitas ataupun langkah dari suatu proses. *Event log* menunjukkan proses tunggal sehingga sebuah kejadian termasuk dalam sebuah proses, yang biasa disebut dengan *case* [5]. *Event log* dapat menyimpan informasi yang berhubungan dengan kejadian, seperti *timestamp* atau *resources* [5].

Timestamp adalah waktu terjadinya suatu kejadian. Sedangkan *resources* adalah pelaku yang mengeksekusi suatu kejadian. Definisi lengkap dari *event log* dapat dilihat pada Gambar 2.2 [5].

Definisi *Event log*

Event log terdiri dari rangkaian aktivitas dan waktu yang didefinisikan dengan $L_{A,TD} = (E, C, \alpha, \gamma, \beta, \succ)$ dimana:

- E adalah kumpulan kejadian.
- C adalah kumpulan case.
- $\alpha: E \rightarrow A$ adalah fungsi yang menghubungkan setiap kejadian dengan sebuah aktifitas.
- $\gamma: E \rightarrow TD$ adalah fungsi yang menghubungkan setiap kejadian dengan sebuah waktu kejadian (*timestamp*).
- $\beta: E \rightarrow C$ adalah fungsi yang menghubungkan sebuah kejadian dengan sebuah case.
- $\succ \subseteq ExE$ adalah *relation succesion*, yang merupakan total suatu kejadian yang dilanjutkan dalam kejadian lain dan termasuk dalam E. $e_2 \succ e_1$ adalah notasi singkat untuk $(e_2, e_1) \in \succ$. Kumpulan kejadian yang berkaitan dalam sebuah case disebut sebagai *trace*.

Gambar 2.2 Definisi *Event Log*

Sedangkan *streaming event log* merupakan *event log*yang diobservasi satu per satu berdasarkan waktu dilaksanakannya kejadian pada *event log* tersebut [5]. Sehingga, *streaming event log* adalah *event log* yang tercatat secara *real-time*.

2.4 *Declarative Miner*

Penemuan model deklaratif secara umum dan model declare secara khusus dapat digunakan untuk secara efektif menyelesaikan dua kelemahan penting dari teknik penemuan proses yang ada:

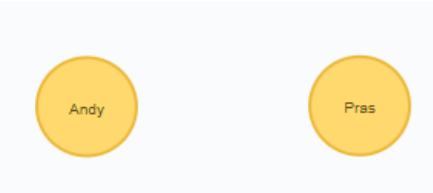
- Model yang dihasilkan cenderung besar dan kompleks terutama di lingkungan yang fleksibel dimana eksekusi proses melibatkan banyak alternatif
- Menawarkan kemungkinan terbatas untuk memandu proses penambangan menuju sifat-sifat menarik yang spesifik

Dengan menggunakan model deklaratif, perilaku proses digambarkan sebagai seperangkat aturan yang harus dipenuhi selama proses eksekusi. Penemuan model deklaratif dapat dengan mudah dipandu dalam kerangka aturan.

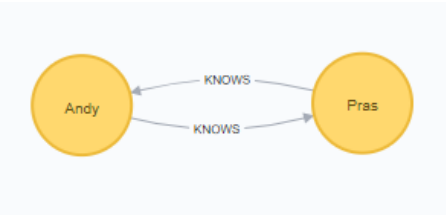
2.5 *Neo4j*

Neo4j adalah aplikasi gratis NoSQL graph database yang di implementasikan di java dan scala dengan pembuatan mulai 2003, dan telah tersedia untuk umum sejak 2007. Neo4j sekarang digunakan oleh ratusan ribu perusahaan dan organisasi di hampir semua industri. Use case yang meliputi matchmaking, manajemen jaringan, analisis perangkat lunak, penelitian ilmiah, routing, manajemen organisasi dan proyek, rekomendasi, jejaring sosial dan lainnya. Neo4j menerapkan model grafik properti secara efisien dalam tingkat penyimpanan.

Neo4j dapat menyimpan 2 hal yang terdiri dari: *Node Label* dan *Relatonship Types*. Node merupakan kumpulan data yang berisi informasi-informasi yang tersimpan dalam sebuah data. Contoh: Sebuah database Person yang menampung informasi seperti Nama dan Asal. Sedangkan Relationship merupakan kumpulan data yang berisi hubungan / relasi antar node. Contoh node dan relasi dapat dilihat di



Gambar 2.3 Contoh Node dalam Neo4J



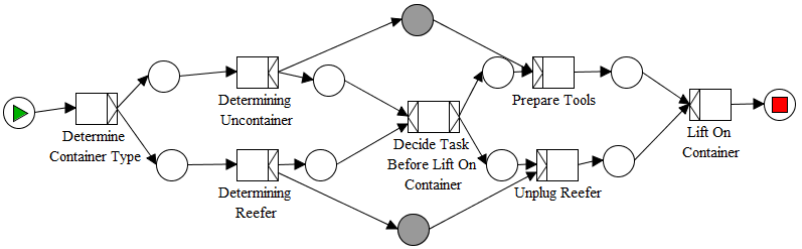
Gambar 2.4 Contoh Relationship dalam Neo4J

2.6 *Non-free choice*

Non-free choice (Pilihan tidak bebas) adalah aktivitas tambahan yang tidak muncul di log data tetapi ditampilkan dalam model proses (Wen, Wang, van der Aalst, Huang, & Sun, 2010). Kegunaan dari *non-free choice* adalah untuk membantu menggambarkan proses secara sebenarnya dalam model proses [7].

Non-free-choice merupakan aktivitas yang kelihatannya memiliki pilihan tetapi sebenarnya tidak ada pilihan. Keuntungan dari *non-free choice* adalah dapat menggambarkan proses bahwa sebuah aktivitas dapat berjalan ketika aktivitas yang berkaitan dengan *non-free choice* telah dijalankan.

Pada **Gambar 2.5**, terdapat kondisi ketika aktivitas *Prepare Tools* dijalankan, maka aktivitas sebelumnya harus sudah menjalankan *Determining Uncontainer* dan ketika aktivitas Unplug Reefer dijalankan, aktivitas *Determining Reefer* harus sudah dijalankan. Dengan kondisi tersebut, maka diperlukan adanya bantuan aktivitas yang disebut *non-free choice* yang ditandakan oleh lingkaran berwarna abu-abu.

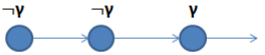
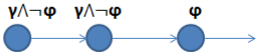


Gambar 2.5 Contoh *non-free-choice*

2.7 Linear Temporal Logic (LTL)

Linear Temporal Logic (LTL) (Raim, 2014) adalah bahasa yang menggambarkan logika temporal yang mengacu pada waktu. Linear Temporal Logic dibangun dari konstanta (benar dan salah), sekelompok proposisi atomic, operator logika (¬, ∨, ∧, →) dan operator modal temporal. Terdapat empat operator modal temporal yang digunakan pada Linear Temporal Logic. Penjelasan mengenai operator tersebut dapat dilihat pada 2.6.

Pada penggalian proses, LTL dibentuk menjadi relasi aktivitas yang disebut *template* Declare (Fabrizio M. Maggi, Mooij, & Van Der Aalst, 2011). Contoh *template* declare dapat dilihat pada **Tabel 2.1**. *Template* ini yang digunakan oleh algoritma *Declare Miner* (Fabrizio Maria Maggi, 2013) untuk membentuk model menggunakan *event log*. Model tersebut berupa graf yang memiliki relasi seperti pada *template Declare*.

Tekstual	Simbol	Penjelasan	Diagram
Xγ	◯γ	γ harus dilaksanakan sebagai *state* selanjutnya	
Gγ	□γ	γ harus berada pada seluruh *state* berikutnya	
Fγ	◊γ	γ harus berada pada tempat lain di dalam *path*.	
γUφ	γUφ	γ harus dilaksanakan sampai φ muncul.	

Gambar 2.6. Operator Modal Temporal (LTL)

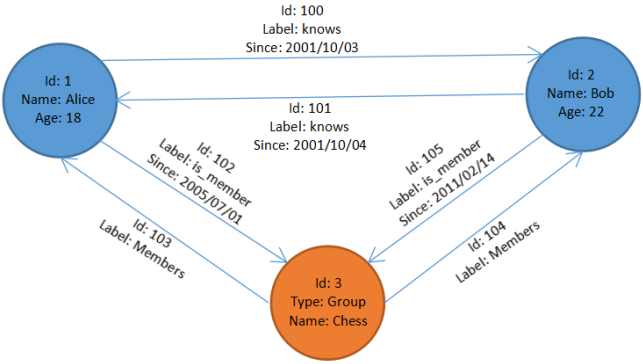
Tabel 2.1 Potongan *Template* Declare

***Template* Declare**	**LTL**	**Deskripsi**
Init(K)	K	K adalah aktivitas pertama yang terekam dalam proses.
Chain Response(K,R)	□ (K → O (R))	Jika K terekam, maka aktivitas yang langsung terekam selanjutnya adalah R
Exclusive Choice (K,R)	(◊ (K ∨ R) ∧ ! (◊ (K ∧ R)))	K atau R akan terekam dalam proses, akan tetapi kedua aktivitas tersebut tidak dapat terekam pada proses yang sama.
Response(K,R)	□ (K → ◊ (R))	Aktivitas R hanya boleh terekam apabila aktivitas K sudah terekam dalam proses

Template **Declare**	**LTL**	**Deskripsi**
Existences2(K)	◊ (K ∧ O (◊ (K)))	Aktivitas K muncul sekurang-kurangnya dua kali di proses

2.8 *Graph Database*

Graph database adalah relasi driven. Melihat struktur data, itu akan menjadi grafik terarah dalam arti matematis. Dibandingkan dengan SQL, mereka sedikit berbeda. Dengan membandingkan deretan data SQL dan entitas dari *database* grafik, dapat dilihat bahwa database grafik memiliki perbedaan yang berbeda. Data akan terdiri dari node dan simpul, seperti yang ditunjukkan pada Gambar 2.7. Simpul akan memiliki semua informasi tentang beberapa objek. Dan simpul akan mewakili hubungan antar objek. Misalnya, id subjek: "1" mengetahui sebuah objek dengan id: "2". Inti dari model data ini adalah untuk mengatasi masalah memiliki terlalu banyak hubungan dalam data tabel. Orang akan berakhir menulis query SQL yang sangat kompleks untuk mendapatkan data dengan banyak tingkat hubungan. Juga menganalisa data seperti grafik sangat alami bagi orang untuk melihat pola relasi, yang akan sulit diperhatikan dalam data tabular



Gambar 2.7 Stuktur Data Graph Database

BAB III
METODE PEMECAHAN MASALAH

Pada bab ini akan dibahas mengenai metodologi pemecahan masalah yang digunakan sebagai dasar solusi dari pembuatan Tugas Akhir. Metodologi tersebut menerangkan langkah demi langkah proses hingga dapat menghasilkan proses model dari suatu *event log*.

3.1 Cakupan Permasalahan

Permasalahan utama yang diangkat pada pembuatan Tugas Akhir ini adalah menemukan proses model yang tepat (*process discovery*) dari *streaming event log* yang memiliki proses yang memiliki hubungan *non-free choice*.

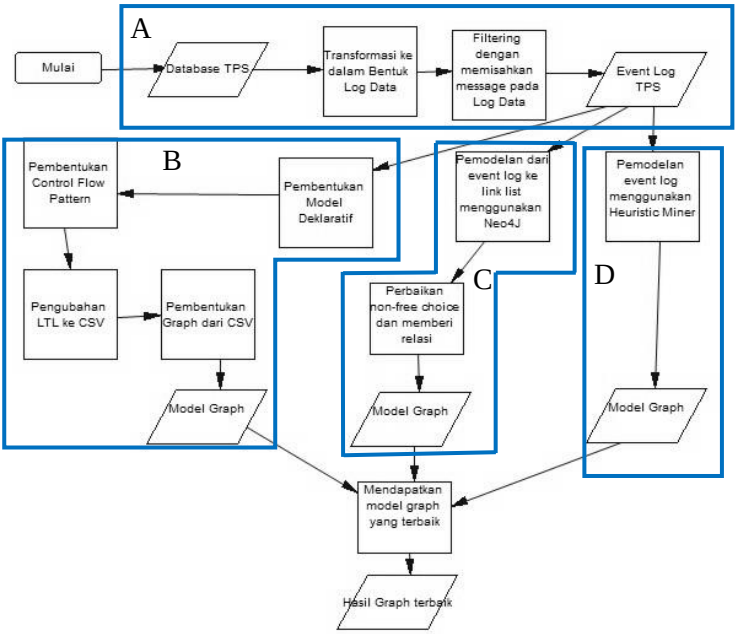
Permasalahan pertama yang dipecahkan dalam Tugas Akhir ini adalah menggambarkan model proses beserta relasinya yang dihasilkan dari Neo4J, *Declarative Miner*, *Heuristic Miner*.

Permasalahan kedua yang dipecahkan pada Tugas Akhir ini adalah memodelkan proses beserta relasinya dengan masukan *event log* yang merupakan sebuah *file* dalam bentuk *comma separated value* (csv).

Permasalahan ketiga yang dipecahkan pada Tugas Akhir ini adalah membandingkan *node* dan relasi antar *graph* yang terbentuk dengan *Declarative Miner, Heuristic Miner* dengan *graph* yang terbentuk langsung dari *event log.*

Gambar 3.1 Alur Proses Pengerjaan Tugas Akhir adalah gambaran umum alur proses pengerjaan Tugas Akhir. Tahapan pertama adalah melakukan transformasi database ke log data serta *filtering* dengan memisahkan *message*. Hal ini dilakukan untuk mendapatkan log data yang dapat diolah dalam algoritma pemodelan proses. Kemudian, log tersebut akan dipecah menjadi 3 log data, yaitu log data bulan Januari, Februari, dan Maret yang kemudian diolah untuk mendapatkan *control-flow patterns*. Pengolahan *control-flow patterns* melibatkan *rules* atau template pada model *Declarative Miner* , Neo4J, dan *Heuristic miner*.

Kemudian, *control-flow patterns* yang telah terbentuk akan disusun ke dalam bentuk graf untuk dibandingkan hasilnya.



Gambar 3.1 Alur Proses Pengerjaan Tugas Akhir

Pada gambar 3.1 dapat dikelompokkan bahwa poin A merupakan alur tentang pre-processing data, dimana data di transformasikan ke *event log*, diolah dengan memisahkan message sebelum di proses ke metode yang kita gunakan. Poin B adalah memodelkan LTL dari *declarative miner*. Dari LTL tersebut didapatkan file CSV yang dapat diolah dalam Neo4j. Poin C adalah memodelkan *event log* ke *link list* dalam Neo4j. Di poin ini kita membuat relasi *non-free choice* menggunakan *Cypher*. Poin D adalah memodelkan *event log* dengan *Heuristic Miner*.

3.2 Masukan

3.2.1 Data

Dalam penelitian ini, log data yang digunakan adalah log data mengenai proses impor barang di Terminal Peti Kemas Surabaya. Log data yang didapatkan merupakan hasil transformasi *database* proses impor barang mulai dari proses yang berjalan dari bulan Januari sampai proses yang berakhir di bulan Maret 2016. Log data yang digunakan mengandung ID_Case (nomor proses), nama aktivitas dan nama *message*, *originator* (pelaku aktivitas), *sender receiver* (pelaku *message*), waktu pelaksanaan, lampiran, detail lampiran, serta biaya pelaksanaan aktivitas.

3.3 Pre-processing

3.3.1 Transformasi ke Log Data dan Pemisahan Log per Bulan

Pada tahap ini dilakukan transformasi dari *database* ke log data. Terdapat dua tahapan dalam transformasi ke log data. Tahapan tersebut adalah penentuan waktu aktivitas serta pemisahan *message* dan aktivitas.

Database proses impor barang TPS menyimpan waktu dari aktivitas dan *message*, akan tetapi tidak semua kolom menyimpan waktu tiap aktivitas dan *message* melainkan terdapat beberapa kolom yang menyimpan waktu beberapa aktivitas serta *message* sekaligus. Akan tetapi, selain *database*, *expert* juga memberikan rentang waktu estimasi pelaksanaan aktivitas dan *message* untuk memberi gambaran waktu pada aktivitas dan *message* yang berkelompok. Dari data-data tersebut, dilakukan pengacakan waktu menggunakan *inverse normal distribution* yang mengacu pada estimasi waktu *expert* dan *database*.

Persamaan untuk perhitungan *inverse normal distribution* dapat dilihat pada Persamaan (1) sampai Persamaan (3). Nilai p yang berarti probabilitas pada Persamaan (1) yang diambil secara random dimana lebih dari 0 dan kurang dari 1. Kemudian, pada Persamaan (2), $\overline{Act_{IN}Log}$ adalah nilai rata-rata kelompok aktivitas di database, $\widetilde{Es.Act}$ adalah median dari rentang waktu estimasi aktivitas, dan $\sum_{i=0}^{n} \widetilde{Es.Act}_i$ adalah jumlah

median rentang waktu seluruh aktivitas yang diwakili kelompok aktivitas di *database*.

$$time(Act) = NormInv(p, \mu, \sigma_x) \quad 1)$$

$$\mu = \frac{\widetilde{Es.Act}}{\sum_{i=0}^{n} \widetilde{Es.Act_i}} \times \overline{ActinLog} \quad 2)$$

$$\sigma_x = 0{,}05 \times \mu \quad 3)$$

Kemudian, pemisahan aktivitas serta *message* didasarkan pada pelaksana. Apabila pelaksana ada dua yaitu *sender* dan *receiver*, maka data itu adalah data *message*. Sedangkan apabila pelaku hanya satu, yang biasa disebut *originator*, maka data tersebut adalah data aktivitas. Pseudocode untuk penentuan *message* dan aktivitas dapat dilihat pada Gambar 3.2. Penambahan aktivitas pada log_aktivitas dan penambahan *message* pada Log_messsage tidak hanya melibatkan nama aktivitas atau *message* saja, melainkan informasi yang mendukung seperti waktu, pelaksana, dan lainnya.

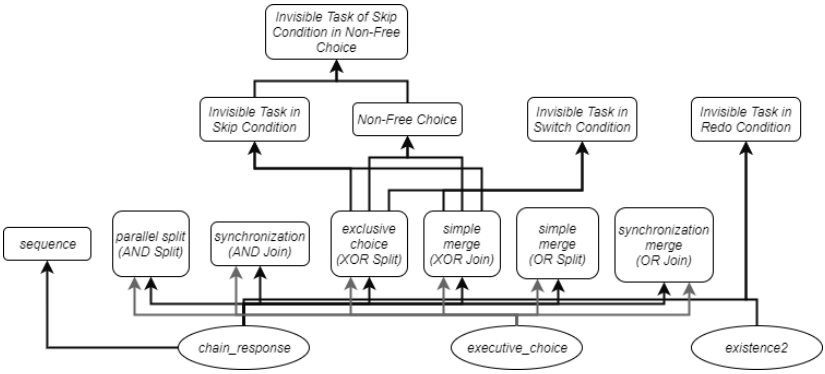
```
.   aktivitas_atau_message = list aktivitas and message
.   for x in aktivitas_atau_message:
.       if sender(x) and receiver(x) != NULL:
.           Log_aktivitas.add(aktivitas)
.       else:
.           Log_message.add(message)
```

Gambar 3.2 Pseudocode pemisahan aktivitas dan *message*

3.4 Pembentukan *Business Process Model*

3.4.1 Pembentukan *Business Process Model* dari Model Deklaratif

Pada penggambaran *control-flow patterns*, model deklaratif yang digunakan menggunakan *rules* atau *template* berupa *chain_response, executive_choice, response,* dan *existence(2)*. Tidak semua *control-flow patterns* ditentukan hanya berdasarkan *rules* pada model deklaratif, melainkan juga berdasarkan *control-flow patterns* lain yang sudah terbentuk. Gambar 3.3 menunjukkan alur pembentukan *control-flow patterns*.



Gambar 3.3 Alur Hirarki Pembentukan *Control-Flow Patterns*

Berdasarkan Gambar 3.3, *Sequence relation* dibentuk dari nilai chain_response, dan relasi lainnya seperti AND Split, AND Join, XOR Split, XOR Join, OR Split, dan OR Join dibentuk dari nilai *chain_response* dan nilai *executive_choice*. Untuk membentuk relasi *non-free choice* dibutuhkan relasi XOR Split dan XOR Join. Metode pembentukan *control-flow patterns* dapat dilihat pada Tabel 3.1. Di dalam metode, terdapat beberapa variabel tambahan. Variabel-variabel tersebut adalah total_after(act), total_before(act), total_ex_after(act), total_ex_before(act), total_chain_after(act), dan total_chain_before(act).

Total_after(act) adalah jumlah *chain_response*(act, aktivitas lainnya) yang memiliki nilai *support* lebih dari 0. Sedangkan total_before(act) adalah jumlah *chain_response*(aktivitas lainnya, act) yang memiliki nilai *support* lebih dari 0. Total_chain_before(act) adalah jumlah *chain_response*(aktivitas sebelum act, aktivitas sebelum act) yang memiliki nilai *support* lebih dari 0. Total_chain_after(act) adalah jumlah *chain_response*(aktivitas setelah act, aktivitas setelah act) yang memiliki nilai *support* lebih dari 0. Total_ex_after(act) adalah jumlah *executive_choice* (aktivitas setelah act, aktivitas setelah act) yang memiliki nilai *support* lebih dari 0 dan rotal_chain_before(act) adalah jumlah *executive_choice*(aktivitas sebelum act, aktivitas sebelum act) yang memiliki nilai *support* lebih dari 0.

Sebagai contoh, *chain_response*(K,R),*chain*_response(K,S), dan *chain_response*(T,S) tergambar dalam model deklaratif. Dari *rules* pada model deklaratif tersebut, total_next(K) dan total_before(S) adalah dua, dan total_next(T) dan total_before(R) adalah satu, serta total_ex_after(K) adalah satu.

Tabel 3.1. Metode untuk membangun control-flow-patterns

Pattern	**Peraturan untuk membangun *control-flow patterns***
Exclusive choice (XOR Split)	`If total_next(act)>1 and total_ex_after(act)>0 and(total_chain_after(act) < total_after(act)):` `act -> O ( ( y1 ∨ y2 …. ∨ yn ) )`

Pattern	**Peraturan untuk membangun *control-flow patterns***
Simple Merge (XOR join)	If total_before(act)>1 and total_ex_before(act)>0 and(total_chain_before(act)< total_next(act)): O ((y1 ∨ y2 …. ∨ yn)) -> O (act)
non-free-choice	For act in all activities: If act in LTLXORSplit as act -> .... and act in LTLXORJoin as ..... -> O (act): If total [executive_choice(x ∈ act_before of act ,y ∈ act_after of act) which has support value > 0] < total(act_before)^2: LTLNonFreeChoice.append(<> (((x1 /\ act /\ k) \/ (x2 /\ act /\ k) \/ …. \/ (xn /\ act /\ k)))), where x1….xn ∈ act_before of act, k = act_next in executive_choice(x, act_next) which has minimum support value delete list that has act in LTLXORSplit and LTLXORJoin

Setelah mendapatkan LTL kemudian kita ubah ke dalam bentuk *cypher* untuk mendapatkan sebuah *graph model*. Kita mencari relasi antar aktivitas juga disini dengan melalui jumlah *incoming relation* dan *outgoing relation* dari aktivitas. Jumlah *incoming relation* adalah jumlah aktivitas yang terjadi sebelumnya, sebaliknya dengan *outgoing relation* adalah jumlah aktivitas yang sebelumnya. Pada tabel 3.2 ditunjukkan bagaimana cara mengubah LTL ke dalam *Cypher*.

Tabel 3.2 Mengubah LTL ke Neo4J

LTL	Cypher
act -> O(y)	*for i=0 until i=count_activity* *create relation activity[i]-[r:NEXT]->activity[i+1]*
act -> O ((y1 ∨ y2 …. ∨ yn))	*match(activity[i])-[r:NEXT]->(activity[i+1])* *if outgoing(activity[i])>1 and incoming(activity[i+1])=1* *create relation activity[i]-[r:XORSPLIT]-> activity [i+1]*
O ((y1 ∨ y2 …. ∨ yn)) -> O (act)	*match(activity[i])-[r:NEXT]->(activity[i+1])* *if outgoing(activity[i])=1 and incoming(activity[i+1])>1* *create relation activity[i]-[r:XORJOIN]-> activity [i+1]*
<>((y1 ∧ act ∧ y2) ∨ (y3 ∧ act ∧ y4))	*match(activity[i])-[r:XORJOIN]-> (activity[i+1])-[r:XORSPLIT]-> (activity[I+2])* *match(activity[i])-[r:SEQUENCE]-> (activity[i+1])-[r:SEQUENCE]-> (activity[I+2])* *where activity[i].name = activity[I].name and activity[I+1].name = activity[I+1].name and*

LTL	Cypher
	activity[I+2].name = activity[I+2].name *create relation activity[i]-[r:NONFREECHOICE]->activity[i+2]*

3.4.2 Pembentukan *Business Process Model* dari Model Neo4J

Untuk mendapatkan sebuah model dari Neo4J, hal pertama yang dilakukan yaitu menginputkan *event log* yang berekstensi CSV ke dalam Neo4J dan dibuat ke dalam *link list*. Cara untuk menginputkan *event log* yang berekstensi CSV ke dalam Neo4J dan dibuat ke dalam *link list* terdapat di Gambar 3.4.

```
//LOAD CSV
LOAD CSV with headers FROM "file e" AS line
Merge (:Activity
{CaseId:line.Case_ID,Name:line.Activity,StartTime:line.Start_Stamp,EndTime:line.End_Stamp })
LOAD CSV with headers FROM "file:///eventlogswitch.csv" AS line
Merge (:CaseActivity {Name:line.Activity })
//CONNECT THE ACTIVITY
Match (c:Activity)
WITH COLLECT(c) AS Caselist
UNWIND RANGE(0,Size(Caselist) - 2) as idx
WITH Caselist[idx] AS s1, Caselist[idx+1] AS s2
match (b:CaseActivity),(a:CaseActivity)
WHERE s1.CaseId = s2.CaseId AND s1.Name = a.Name  AND s2.Name = b.Name
MERGE (a)-[r:NEXT]->(b)
```

Gambar 3.4 Memasukkan event log ke Neo4J

Setelah terbentuk ke dalam *link list*, maka kita mencari relasi antar aktivitas, mengecek *event log* apakah mengandung *non-free choice* atau tidak. Jika ada maka kita memperbaiki *graph model*. Cara untuk mencari relasi ialah dengan jumlah *incoming relation* dan *outgoing relation* dari aktivitas. Jumlah *Incoming relation* adalah Jumlah aktivitas yang terjadi sebelumnya, sebaliknya dengan *outgoing relation* adalah jumlah aktivitas yang sebelumnya. Pada tabel 3.3 dapat dilihan metode *cypher* untuk menemukan sebuah pattern.

Tabel 3.3 Penemuan Pola dalam Neo4J

Pattern	Cypher
Sequence	*for i=0 until i=count_activity* *create relation activity[i]-[r:NEXT]->activity[i+1]*
Exclusive choice (XOR Split)	*match(activity[i])-[r:NEXT]->(activity[i+1])* *if outgoing(activity[i])>1 and incoming(activity[i+1])=1* *create relation activity[i]-[r:XORSPLIT]-> activity [i+1]*
Simple Merge (XOR join)	*match(activity[i])-[r:NEXT]->(activity[i+1])* *if outgoing(activity[i])=1 and incoming(activity[i+1])>1* *create relation activity[i]-[r:XORJOIN]-> activity [i+1]*
non-free-choice	match(activity[i])-[r:XORJOIN]-> (activity[i+1])-[s:XORSPLIT]-> (activity[I+2]) match(activity[i])-[:SEQUENCE]-> (activity[i+1])-[:SEQUENCE]-> (activity[I+2]) where activity[i].name = activity[I].name and activity[I+1].name = activity[I+1].name and activity[I+2].name = activity[I+2].name

Pattern	Cypher
	create relation activity[i]-[r:NONFREECHOICE]->activity[i+2] delete r,s

3.4.3 Pembentukan *Business Process Model* dari *Model Heuristic Miner*

Kita akan membentuk *Business Process Model* dari Model *Heuristic Miner* menggunakan ProM dan Disco. Pertama-tama kita akan menggunakan Disco untuk mengubah *event log* dari berekstensi CSV menjadi mxml dimana ekstensi mxml ini akan digunakan sebagai input di ProM. Di ProM setelah menginputkan *event log* berekstensi mxml pilih start analyzing kemudian pilih *Heuristic Miner*. Setelah itu, kita akan disuruh untuk mengisi nilai dari *Relative to best threshold, Positive observations, Dependency threshold, Length one loops threshold, length two loops threshold, long distance threshold, Dependency divisor, dan And threshold*, tapi kita akan memakai nilai *default* semua. Setelah semua nilai terisi, maka kita melakukan *start analyzing*.

3.5 Keluaran

3.5.1 Metode Pengujian

Pada hasil dari tugas akhir ini terdapat 3 *graph* yang menggambarkan *business process model* yaitu *graph* yang berasal dari LTL, *graph* yang berasal dari Neo4J, dan *graph* yang berasal dari *Heuristic Miner*. Untuk menguji baik buruk nya metode *graph database* ini, maka kita mengkomparasi antara 3 *graph* ini dan mengetahui *graph* mana yang merupakan paling baik.

3.5.2 Skenario Uji Coba

Skenario uji coba berdasarkan metode pengujian yang dijelaskan pada bab 3.5.1. Tedapat beberapa pengujian yang dilakukan, yaitu :

1. Menguji apakah *event log* yang terbentuk dari *database* sudah terbentuk dengan benar.
2. Menguji apakah *event log* yang terbentuk dari *Declarative Miner* sudah meenangani *non-free choice* dengan benar.
3. Menguji apakah *event log* yang terbentuk dari *Neo4J* sudah menangani *non-free choice* dengan benar.
4. Menguji apakah *event log* yang terbentuk dari *Heuristic Miner* sudah meenangani *non-free choice* dengan benar.
5. Membandingkan metode manakah yang terbaik untuk memodelkan *Business Process Model*.

BAB IV
IMPLEMENTASI

Bab ini membahas tentang implementasi dari perancangan sistem. Bahasa pemrograman yang digunakan adalah bahasa pemrograman Java.

4.1 Lingkungan Implementasi

Lingkungan implementasi merupakan lingkungan dimana sistem ini dibangun, dimana akan dijelaskan mengenai kebutuhan perangkat yang diperlukan untuk membangun sistem ini. Lingkungan implementasi dibagi menjadi dua, yaitu lingkungan implementasi terhadap perangkat keras dan lingkungan implementasi terhadap perangkat lunak.

4.1.1. Perangkat Keras

Implementasi dilakukan pada sebuah laptop dengan spesifikasi sebagai berikut:

- Merk : ASUS.
- Seri : A Series 555L.
- Prosesor : Intel Core i5 @ 2.20 GHz.
- RAM : 8 GB.

4.1.2. Perangkat Lunak

Perangkat lunak yang mendukung fungsionalitas sistem adalah:

1. Sistem Operasi Windows
   Sistem operasi yang digunakan adalah Microsoft Windows 7 64-bit.
2. Java
3. Eclipse
4. ProM
5. Neo4j
6. SQL

4.2 Penjelasan Implementasi

Pada sub bab ini dijelaskan implementasi setiap metode yang dijelaskan pada bab metode pemecahan masalah, sehingga terbentuk suatu perangkat lunak yang mengimplementasi metode-metode pada bab metode pemecahan masalah.

4.2.1 Implementasi

Sub bab ini membahas implementasi tampilan antarmuka pengguna pada Java serta penghubung antar tampilan. Berikut ini dijelaskan mengenai implementasi tampilan antarmuka pengguna yang terdapat pada perangkat lunak.

4.2.1.1 Pre-Processing

Antarmuka Awal Menjalankan Simulasi *Process Discovery* berfungsi sebagai perantara sistem dan pengguna dalam melakukan pemilihan *event log* sebagai data simulasi untuk *process discovery* menggunakan algoritma yang diusulkan. Terdapat beberapa fungsi yang digunakan sebagai perantara tersebut.

4.2.1.1.1 Implementasi Transformasi ke Log Data

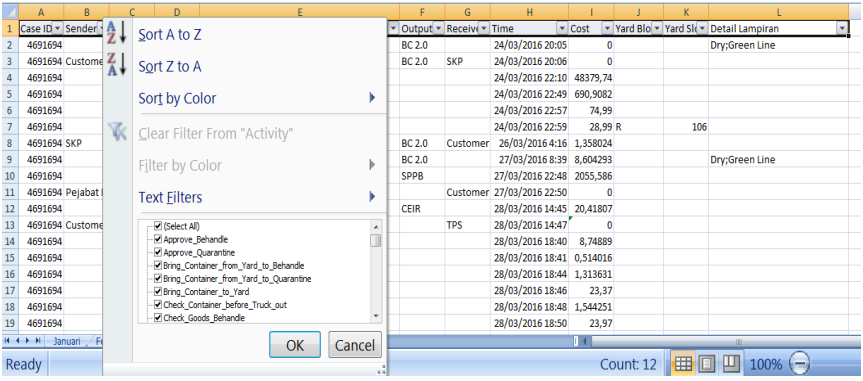
Untuk melakukan pre-processing data, langkah pertama yang harus dilakukan adalah transformasi dari *database* menjadi log data. Disini kita mengimplementasikan metode yang telah dijabarkan di bab 3.3.1. Untuk mengubah data dari database ke event log maka perlu melakukan yang ada di Kode Sumber 4.1. Kode Sumber 4.1merupakan bagian dari source code bukan semuanya. Disini kita hanya perlu mengganti variabel-variabel yang akan diubah beserta nilainya, dan kalau variabel waktu terdapat perhitungan distribusi terlebih dahulu

```
INSERT INTO `table 4`
(Case_id,Sender,Originator,Input,Activity,Output,Receiver,actTime,Cost,Yard_block,Yard_slot,De
ail_lampiran)
SELECT CONTAINER_KEY as Case_id,'' as Sender,'Customer' as Originator, 'NPWP, SIUP, API, SRP,
TDP, NPIK, IT, INVOICE, PO, SK, BL, COO' as Input,'Document_Entry_via_PDE' as Activity,  'BC
2.0' as Output,'' as Receiver,( VESSEL_ATB-((RAND()*(0.125-0.08333333))+0.08333333) ) AS
Time,'0' as Cost,'' as Yard_block,'' as Yard_slot,
(IF(HAS_QUARANTINE_FLAG ='YES',
(case when CTR_TYPE='DRY' && CUSTOMS_BEHANDLE_COUNT=0 then 'Quarantine;Dry;Green Line'
          when CTR_TYPE='DRY' && CUSTOMS_BEHANDLE_COUNT=1 then 'Quarantine;Dry;Red Line'
          when CTR_TYPE='RFR' && CUSTOMS_BEHANDLE_COUNT=0 then 'Quarantine;Reefer;Green Line'
          when CTR_TYPE='RFR' && CUSTOMS_BEHANDLE_COUNT=1 then 'Quarantine;Reefer;Red Line'
          when CTR_TYPE='OVD' && CUSTOMS_BEHANDLE_COUNT=0 then 'Quarantine;Uncontainer;Green
        Line'
          when CTR_TYPE='OVD' && CUSTOMS_BEHANDLE_COUNT=1 then 'Quarantine;Uncontainer;Red
Line'
          ELSE ''
          END
) ,(case when CTR_TYPE='DRY' && CUSTOMS_BEHANDLE_COUNT=0 then 'Dry;Green Line'
          when CTR_TYPE='DRY' && CUSTOMS_BEHANDLE_COUNT=1 then 'Dry;Red Line'
          when CTR_TYPE='RFR' && CUSTOMS_BEHANDLE_COUNT=0 then 'Reefer;Green Line'
          when CTR_TYPE='RFR' && CUSTOMS_BEHANDLE_COUNT=1 then 'Reefer;Red Line'
          when CTR_TYPE='OVD' && CUSTOMS_BEHANDLE_COUNT=0 then 'Uncontainer;Green Line'
          when CTR_TYPE='OVD' && CUSTOMS_BEHANDLE_COUNT=1 then 'Uncontainer;Red Line'
          ELSE ''
          END
) )) as Detail_lampiran
```

Kode Sumber 4.1 Transformasi Data

4.2.1.1.2 Implementasi Pemisahan Message dari Log Data serta Pengelompokan Data per Bulan

Setelah mendapatkan *event log* dalam bentuk excel, maka kita perlu memilah data dengan kondisi mana aktivitas mana *message*. Jika *message* maka akan dibuang. Cara untuk implementasi dilakukan melalui fitur filter excel pada Kode Sumber 4.2



Kode Sumber 4.2 Filter data dari Message

4.2.1.2 Implementasi Pembentukan Business Process Model

Kita perlu untuk mengimplementasi pembentukan *Business Process Model*. Pengimplementasian pembentukan *business process model* dibagi menjadi 3 yaitu pembentukan *business process model* dari *Declarative model*, Neo4J, dan *Heuristic Miner*

4.2.1.2.1 Implementasi Pembentukan Business Process Model dari Model Deklaratif

Untuk melakukan pembentukan Business Process Model dari Model Deklaratif pertama harus install plugin ProM ke Eclipse. Terus modifikasi kode sumber nya di kelas declare miner seperti di bawah. Beberapa variabel yang digunakan yaitu:

total_after : jumlah dari chain_response(act, aktivitas setelah act)
total_before : jumlah dari chain_response(aktivitas sebelum act, act)
totalCRnext : jumlah dari chain_response(aktivitas setelah act, act)
totalCRbef : jumlah dari chain_response(act, aktivitas sebelum act)
totalCRsplit : jumlah dari chain_response(aktivitas setelah act, aktivitas setelah act)
totalCRjoin : jumlah dari chain_response(aktivitas sebelum act, aktivitas sebelum act)
totalEXsplit : jumlah dari exclusive_choice(aktivitas setelah act, aktivitas setelah act)
totalEXjoin : jumlah dari exclusive_choice(aktivitas sebelum act, aktivitas sebelum act)

Untuk mendapatkan *sequence relation* maka cari *Chain_Response* dari (act, y) sesuai pada Kode Sumber 4.3

```
if(declareMinerOutput.getConstraintParametersMap().get(j).get(0).equals(act)
&&declareMinerOutput.getTemplate().get(j).equals(DeclareTemplate.Chain_Re
sponse)){
   y.add(declareMinerOutput.getConstraintParametersMap().get(j).get(1));
}
if(total_after.get(act)==1){
   LTLtemp.add(act);
```

```
    LTLtemp.add(y.get(0));
    LTLSequence.add(LTLtemp);
}
if(total_after.get(act)==0 &&total_after_resp.get(act)==0){
    LastAct = act;
} elseif(total_before.get(act)==0 &&total_before_resp.get(act)==0){
    FirstAct = act;
}
```

Kode Sumber 4.3. Mendapatkan *Sequence Relation*

Kode sumber pada bagian ini untuk mendapatkan semua *control-flow pattern* bagian *split* (AND *split*, XOR *split*, OR *split*)
Untuk mendapatkan relasi AND *Split*, maka cek total_after dari act dan totalCRNext apabila sesuai maka simpan aktivitas pada LTLOtherSplit seperti pada Kode Sumber 4.4

```
if(total_after.get(act)>1 && totalCRnext==0){
  LTLtemp.add(act);
  if(totalEXsplit==0){
    for(int j=0;j<y.size();j++){
      LTLtemp.add(y.get(j));
    }
    LTLOtherSplit.add(LTLtemp);
}
```

Kode Sumber 4.4. Mendapatkan relasi AND *Split*

Untuk mendapatkan relasi XOR *split*, maka cek apakah totalCRnext kurang dari total_after act, bila terpenuhi maka simpan aktivitas pada LTLXORSplit. Implementasi bisa dilihat pada Kode Sumber 4.5

```
  else {
    if(totalCRnext<total_after.get(act)){
      for(int j=0;j<y.size();j++){
        LTLtemp.add(y.get(j));
      }
      LTLXORSplit.add(LTLtemp);
```

Kode Sumber 4.5. Mendapatkan relasi XOR *Split*

Untuk mendapatkan relasi OR split, maka dengan kondisi else dari kedua kondisi di atas, lalu aktivitas tersebut disimpan dalam variabel LTLOtherSplit2. Implementasi ada pada Kode Sumber 4.6

```
    } else {
      for(int j=0;j<y.size();j++){
        LTLtemp.add(y.get(j));
      }
      LTLOtherSplit2.add(LTLtemp);
    }
  }
}
```

Kode Sumber 4.6. Mendapatkan relasi OR *split*

Kode sumber pada bagian ini untuk mendapatkan semua *control-flow pattern* bagian *join* (AND join, XOR join, OR join)
Untuk mendapatkan AND join, maka cek total_before dari act dan totalCRNext. Lalu cek totalEXjoin. Ketika terpenuhi, maka hapus *sequence relation* yang mengarah ke aktivitas act dan tambahkan aktivitas tersebut ke dalam LTLOtherJoin. Implementasi dapat dilihat pada Kode Sumber 4.7

```
if(total_before.get(act)>1 && totalCRnext==0){
  LTLtemp.add(act);
  if(totalEXjoin==0){
    for(int j=0;j<x.size();j++){
      LTLtemp.add(x.get(j));
      for(int k=0;k<LTLSequence.size();k++){
        if(LTLSequence.get(k).get(1).equals(act) &&
          LTLSequence.get(k).get(0).equals(x.get(j))){
            LTLSequence.remove(k);
            break;
          }
        }
      }
      LTLOtherJoin.add(LTLtemp);
    }
```

Kode Sumber 4.7. Mendapatkan relasi AND *join*

Untuk mendapatkan relasi XOR *join* maka cek totalCRbef dan total_before dari act. Jika kondisi terpenuhi, hapus relasi *Sequence*

yang mengarah ke act. Lalu simpan aktivitas pada variabel XOR *join.* Implementasi dapat dilihat pada Kode Sumber 4.8

```
else{
  if(totalCRbef<total_before.get(act)){
    for(int j=0;j<x.size();j++){
      LTLtemp.add(x.get(j));
      for(int k=0;k<LTLSequence.size();k++){
         if(LTLSequence.get(k).get(1).equals(act) &&
           LTLSequence.get(k).get(0).equals(x.get(j))){
             LTLSequence.remove(k);
             break;
           }
         }
      }
      LTLXORJoin.add(LTLtemp);
  }
}
```

Kode Sumber 4.8. Mendapatkan relasi XOR join

Untuk mendapatkan relasi OR *join* yaitu dengan menggunakan kondisi *else* dari kedua kondisi *join* lainnya lalu simpan aktivitas pada variabel LTLOtherJoin2. Implementasi dapat dilihat pada Kode Sumber 4.9.

```
else {
  for(int j=0;j<x.size();j++){
    LTLtemp.add(x.get(j));
    for(int k=0;k<LTLSequence.size();k++){
         if(LTLSequence.get(k).get(1).equals(act) &&
           LTLSequence.get(k).get(0).equals(x.get(j))){
             LTLSequence.remove(k);
             break;
           }
         }
       }
    LTLOtherJoin2.add(LTLtemp);
}
```

Kode Sumber 4.9. Mendapatkan relasi OR *Join*

Mengecek apakah act ada pada LTLXORSplit sebagai act -> … dan act ada pada LTLXORJoin sebagai … -> act seperti pada Kode Sumber 4.10.

```
for(int jj=0;jj<LTLXORSplit.size();jj++){
for(int kk=0;kk<LTLXORJoin.size();kk++){
if(LTLXORSplit.get(jj).get(0).equals(act)&&
LTLXORJoin.get(kk).get(0).equals(act)){
```

Kode Sumber 4.10. Mengecek aktivitas untuk *non-free choice*

Variabel totalEXnfc merupakan jumlah dari exclusive_choice (aktivitas sebelum act, aktivitas setelah act). Lalu cari nilai *support* terendah untuk mendapatkan relasi *non-free choice.* Lalu aktivitas tersebut disimpan dalam LTLNonFreeChoice sesuai pada implementasi yang dapat dilihat pada Kode Sumber 4.11.

```
if(totalEXnfc<x.size()*x.size()){
for(int j=0;j<x.size();j++){
LTLtemp = new ArrayList<String>();
float minSupp = 2;
int index=0;
for(int k=0;k<y.size();k++){
boolean flag = false;
for(int l=1;l<=declareMinerOutput.getConstraintParametersMap().size();l++){
if(declareMinerOutput.getConstraintParametersMap().get(l).get(1).equals(y.get(
k)) &&
declareMinerOutput.getConstraintParametersMap().get(l).get(0).equals(x.get(j))
&&
declareMinerOutput.getTemplate().get(l).equals(DeclareTemplate.Exclusive_Ch
oice) ||
declareMinerOutput.getConstraintParametersMap().get(l).get(0).equals(y.get(k))
&&
declareMinerOutput.getConstraintParametersMap().get(l).get(1).equals(x.get(j))
&&
declareMinerOutput.getTemplate().get(l).equals(DeclareTemplate.Exclusive_Ch
oice)){
flag = true;
if(declareMinerOutput.getSupportRule().get(l)<minSupp){
minSupp = declareMinerOutput.getSupportRule().get(l);
index = l;
}
}
}
if(!flag){
minSupp = 0;
awal = x.get(j);
```

```
akhir = y.get(k);
}
}
LTLtemp.add(act);
if(minSupp==0){
LTLtemp.add(awal);
LTLtemp.add(akhir);
} else {
LTLtemp.add(declareMinerOutput.getConstraintParametersMap().get(index).get
(0));
LTLtemp.add(declareMinerOutput.getConstraintParametersMap().get(index).get
(1));
}
LTLNonFreeChoice.add(LTLtemp);
}
LTLXORSplit.remove(jj);
LTLXORJoin.remove(kk);
break;
}
}
}
}
}
```

Kode Sumber 4.11. Mendapatkan relasi *non-free choice*

4.2.1.2.2 Implementasi Pembentukan Business Process Model dari Model Neo4J

Untuk mengimplementasikan pembentukan *Business Process model* dari model Neo4J, langkah pertama kita harus membuat *link list* dari *event log* yang berbentuk CSV. Untuk kode sumber nya ada di Kode Sumber 4.12.

```
//Load file to node activity
LOAD CSV with headers FROM "file:///eventlogtps.csv" AS line
Merge (:Activity{CaseId:line.Case_ID,Name:line.Activity,StartTime:line.Start_Stamp,EndTime:line.End_Stamp })

//Load file to node CaseActivity
LOAD CSV with headers FROM "file:///eventlogtps.csv" AS line
Merge (:CaseActivity {Name:line.Activity })

//Create next relation
Match (c:Activity)
WITH COLLECT(c) AS Caselist
UNWIND RANGE(0,Size(Caselist) - 2) as idx
WITH Caselist[idx] AS s1, Caselist[idx+1] AS s2
match (b:CaseActivity),(a:CaseActivity)
WHERE s1.CaseId = s2.CaseId AND s1.Name = a.Name  AND s2.Name = b.Name
MERGE (a)-[r:NEXT]->(b)
```

Kode Sumber 4.12 Mengubah *event log* ke *link list*

Setelah berhasil mendapatkan *link list*, maka kita akan mendeteksi relasi antar aktivitas dengan melihat *outgoing relation* dan *incoming relation* seperti pada Kode Sumber 4.13.

```
match (n)-[r:NEXT]->(a)
where size((n)-->())>1 and (size((a)<--()) = 1 and size((a)-->()) = 1)
Merge(n)-[:XORSplit]->(a)
Return distinct 'XORSplit', n.Name, a.Name

match (n)-[r:NEXT]->(a)
where size((n)-->())=1 and (size((a)<--()) > 1)
Merge(n)-[:XORJoin]->(a)
Return distinct 'XORJoin', n.Name, a.Name
```

Kode Sumber 4.13 Membentuk Relasi di Neo4J

Setelah mengetahui relasi maka kita mengecek *non-free choice* dari *graph model* tersebut seperti pada Kode Sumber 4.14.

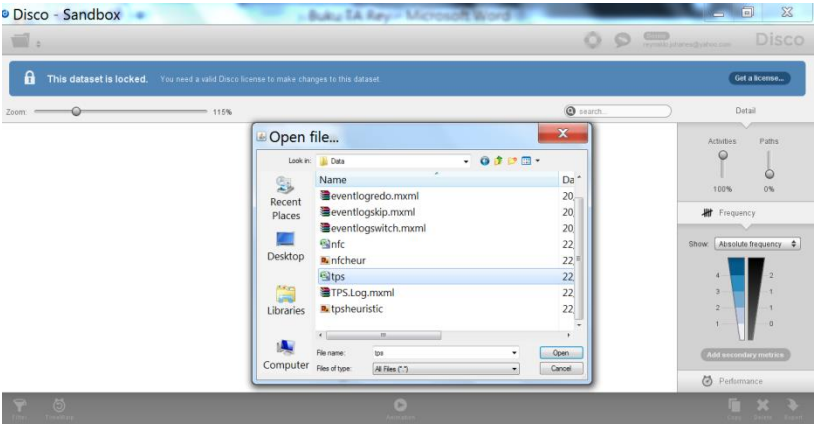
```
//Load CSV and make sequence relation
LOAD CSV WITH HEADERS FROM "file:///eventlogtps.csv" AS line
MERGE (n:SeqActivity {
CaseId:line.Case_ID,
Name:line.Activity,
StartTime:line.Start_Stamp,
EndTime:line.End_Stamp
})
WITH n ORDER BY n.Start_Stamp ASC
WITH COLLECT(n) AS activities
FOREACH (m IN RANGE(0, SIZE(activities)-2) |
FOREACH (prec IN [activities[m]] |
FOREACH (next IN [activities[m+1]] |
MERGE (prec)-[:SEQUENCE]->(next))))

//Check NonFreeChoice
match p1=(a)-[r:XORJOIN]->(n), p2=(n)-[s:XORSPLIT]->(b)
match q1=(x)-[:SEQUENCE]->(y), q2=(y)-[:SEQUENCE]->(z)
where x.Name=a.Name and y.Name=n.Name and z.Name=b.Name
create (a)-[:NONFREECHOICE]->(b)
delete r,s
```

Kode Sumber 4.14 Mengecek *non-free choice*

4.2.1.2.3 Implementasi Pembentukan *Business Process Model* dari *Model Heuristic Miner*

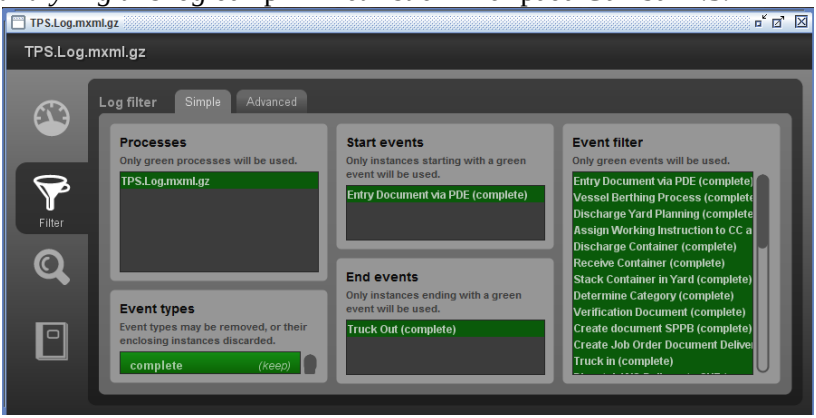
Untuk memodelkan pembentukan *Business Process Model* dari *Model Heuristic Miner* maka langkah pertama yang harus kita lakukan adalah membuat ekstensi mxml dari *event log*.

Buka Disco serta masukkan *event log* ke dalam disco serta *export* ke mxml dari disco seperti di Gambar 4.1.

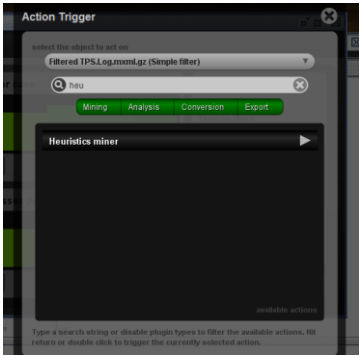


Gambar 4.1 Memasukkan Event Log ke Disco

Setelah mendapatkan ekstensi mxml kita masukkan ke dalam ProM seperti pada Gambar 4.2. Lalu kita pilih *start analyzing this log* dan pilih *Heuristic Miner* pada Gambar 4.3.

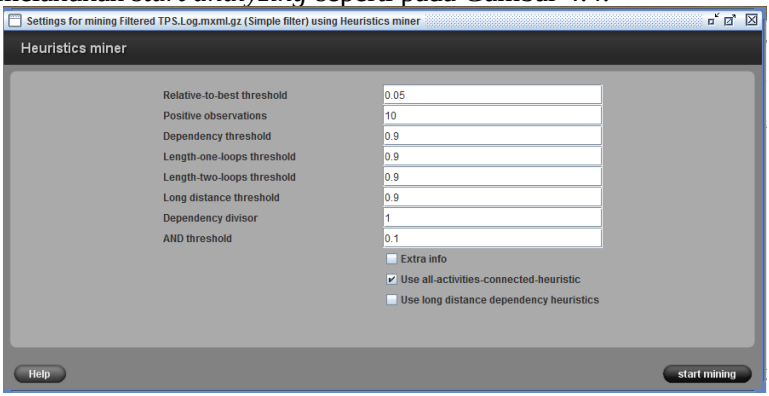


Gambar 4.2 Memasukkan *Event Log* ke ProM



Gambar 4.3 *Heuristic Miner*

Setelah itu kita harus mengisi nilai dari *Relative to best threshold, Positive observations, Dependency threshold, Length one loops threshold, length two loops threshold, long distance threshold, Dependency divisor, dan And threshold*, tapi kita akan memakai nilai *default* semua. Setelah semua nilai terisi, maka kita melakukan *start analyzing* seperti pada Gambar 4.4.



Gambar 4.4 Memasukkan nilai untuk melakukan *Heuristic Miner*

BAB V
PENGUJIAN DAN EVALUASI

Bab ini membahas hasil dan pembahasan pada aplikasi yang dikembangkan. Pada bab ini akan dijelaskan tentang data yang digunakan, hasil yang didapatkan dari penggunaan perangkat lunak dan uji coba yang dilakukan pada perangkat lunak yang telah dikerjakan untuk menguji apakah fungsionalitas aplikasi telah diimplementasikan dengan benar dan berjalan sebagaimana mestinya. Skenario uji coba berdasarkan metode pengujian yang dilakukan, yaitu:

1. Menguji apakah *event log* yang terbentuk dari *database* sudah terbentuk dengan benar.
2. Menguji apakah *event log* yang terbentuk dari *Declarative Miner* sudah meenangani *non-free choice* dengan benar.
3. Menguji apakah *event log* yang terbentuk dari *Neo4J* sudah menangani *non-free choice* dengan benar.
4. Menguji apakah *event log* yang terbentuk dari *Heuristic Miner* sudah meenangani *non-free choice* dengan benar.
5. Membandingkan metode manakah yang terbaik untuk memodelkan *Business Process Model*.

5.1 Lingkungan Uji Coba

Lingkungan uji coba menjelaskan lingkungan yang digunakan untuk menguji implementasi pembuatan sistem pada tugas akhir ini. Lingkungan uji coba meliputi perangkat keras dan perangkat lunak yang dijelaskan sebagai berikut:

1. Perangkat keras
   a. Prosesor : AMD A8-5545 M APU @ 1.70GHz.
   b. Memori(RAM) : 8 GB.
   c. Tipe sistem : 64-bit sistem operasi.
2. Perangkat lunak
   a. Sistem operasi : Windows 8.1 Single Languange.
   b. Kakas bantu : Java, Neo4J, ProM, Eclipse, SQL.

5.2 Pre-Processing

5.2.1 Hasil Transformasi ke Log Data

Database TPS yang digunakan dapat dilihat pada Gambar 5.1. Contoh kolom yang diolah bukan sebagai waktu dari aktivitas atau *message* adalah kolom Container_Key dan Container_Type. Kolom paling kiri adalah kolom Container_Key yang digunakan sebagai Case_ID dan kolom Container_Type akan digunakan sebagai atribut yang disebut sebagai detail lampiran pada log data. Kemudian, log data tersebut diolah menggunakan metode yang dijelaskan pada Sub Bab 3.3.1. Hasil dari pengolahan log data dapat dilihat pada Gambar 5.2. Pada log data yang tergambar di Gambar 5.2, *message* maupun aktivitas masih terekam menjadi satu di kolom nama aktivitas.

5.2.2 Hasil Pemisahan Message dari Log Data serta Pengelompokan Data per Bulan

Agar dapat digunakan untuk pemodelan, log data di-*filter* terlebih dahulu dengan memisahkan antara *message* dan aktivitas. Gambar 5.3 menunjukkan log data yang sudah di-*filter* dari *message*. Jika dilihat pada Log yang sudah di-*filter*, terdapat beberapa aktivitas seperti *Request Behandle* dan *Approve Behandle* terhapus dari log. Itu menunjukkan bahwa aktivitas-aktivitas tersebut merupakan *message*. Kemudian, log yang sudah di-*filter* tersebut akan dikelompokkan menjadi Log Data Bulan Januari, Log Data Bulan Februari, dan Log Data Bulan Maret sebagai bahan uji coba pembentukan model dan perbaikan pada proses yang terpotong.

CONTA	CTR_SIZ	CTR_TY	GROSS	VESSEL_ATB	DISC_DATE	YARD_I	YARD_S	STACK_DATE
4484646	40	DRY	29.103	1/31/16 14:35	2/1/16 20:22	M	112	2/1/16 21:00
4484654	40	DRY	28.408	1/31/16 14:35	2/1/16 20:24	I	126	2/1/16 21:14
4485330	20	DRY	24.82	1/31/16 14:35	2/1/16 21:52	S	13	2/1/16 22:13
4484680	20	DRY	22.235	1/31/16 14:35	2/1/16 21:54	S	13	2/1/16 22:17
4484670	20	DRY	23.066	1/31/16 14:35	2/1/16 21:55	S	13	2/1/16 22:18
4484683	20	DRY	23.085	1/31/16 14:35	2/1/16 22:04	S	13	2/1/16 22:22
4484679	20	DRY	27.1	1/31/16 14:35	2/1/16 22:06	S	13	2/1/16 22:24
4484669	20	DRY	22.707	1/31/16 14:35	2/1/16 22:08	S	29	2/1/16 22:26
4484663	20	DRY	13.608	1/31/16 14:35	2/1/16 22:11	S	29	2/1/16 22:27
4484862	20	DRY	29.245	1/31/16 14:35	2/1/16 1:08	M	27	2/1/16 1:22
4485109	40	DRY	23.29	1/31/16 14:35	2/1/16 1:09	M	30	2/1/16 1:30
4484857	20	DRY	23.088	1/31/16 14:35	2/1/16 1:10	M	27	2/1/16 1:23

Gambar 5.1 Database Proses Impor Barang Terminal Peti Kemas

Case ID	Sender	Origina	Input	Activity	Output	Receiv	Time	Cost	Yar	Yar	Detail Lampira
4453421		Customer	NPWP, SI	Document_Entry_via_PDE	BC 2.0		23/01/2016 7:22	0			Dry;Green Line
4453421	Customer		BC 2.0	Request_Behandle	BC 2.0	SKP	23/01/2016 7:23	0			
4453421		TPS		Vessel_Berthing_Process			23/01/2016 10:10	47836,7172			
4453421		TPS		Discharge_Container			23/01/2016 20:32	428,1331788			
4453421		TPS		Bring_Container_to_Yard			23/01/2016 20:51	50			
4453421		TPS		Stack_Container_in_Yard			23/01/2016 20:54	19,34	J	61	
4453421	SKP		BC 2.0	Approve_Behandle	BC 2.0	Customer	24/01/2016 11:07	1,467092317			
4453421		SKP	BC 2.0	Verification_Document_Beha	BC 2.0		25/01/2016 0:55	6,210446525			Dry;Green Line
4453421		Pejabat B	LHP, BC 2.	Create_document_SPPB	SPPB		25/01/2016 7:48	3450,546591			
4453421	Pejabat Bea	Cukai	SPPB	Send_SPPB_Info		Customer	25/01/2016 7:48	0			
4453421		Customer	SPPB	Create_Job_Order_Documen	CEIR		25/01/2016 15:33	45,79705642			
4453421	Customer			Send_Job_Order_Delivery_Info		TPS	25/01/2016 15:35	0			
4453421		Customer		Truck_in			27/01/2016 15:34	11,00157762			
4453421		TPS		Dispatch_WQ_Delivery_to_CHE			27/01/2016 15:35	1,330475647			
4453421		TPS		Determine_Container_Type			27/01/2016 15:38	1,810399109			
4453421		TPS		Determining_Dry			27/01/2016 15:40	14,79			

Gambar 5.2 Log Data Proses Impor BarangTerminal Peti Kemas

Case ID	Sender	Originator	Input	Activity	Output	Receiver	Time	Cost	Yard I	Yard	Detail Lampiran
4453421		Customer	NPWP, SI	Document_Entry_via_PDE	BC 2.0		23/01/2016 7:22	0			Dry;Green Line
4453421		TPS		Vessel_Berthing_Process			23/01/2016 10:10	47836,72			
4453421		TPS		Discharge_Container			23/01/2016 20:32	428,1332			
4453421		TPS		Bring_Container_to_Yard			23/01/2016 20:51	50			
4453421		TPS		Stack_Container_in_Yard			23/01/2016 20:54	19,34	J	61	
4453421		SKP	BC 2.0	Verification_Document_Behandle	BC 2.0		25/01/2016 0:55	6,210447			Dry;Green Line
4453421		Pejabat Be	LHP, BC 2.	Create_document_SPPB	SPPB		25/01/2016 7:48	3450,547			
4453421		Customer	SPPB	Create_Job_Order_Document_Deli	CEIR		25/01/2016 15:33	45,79706			
4453421		Customer		Truck_in			27/01/2016 15:34	11,00158			
4453421		TPS		Dispatch_WQ_Delivery_to_CHE			27/01/2016 15:35	1,330476			
4453421		TPS		Determine_Container_Type			27/01/2016 15:38	1,810399			
4453421		TPS		Determining_Dry			27/01/2016 15:40	14,79			
4453421		TPS		Decide_Task_Before_Lift_Container			27/01/2016 15:42	3,166661			
4453421		TPS		Lift_on_Container_Truck			27/01/2016 15:44	15,98			
4453421		Customer		Truck_Go_To_Gate_Out			27/01/2016 16:30	1,992126			

Gambar 5.3 Log Data Tanpa *Message* pada Proses Impor Barang Terminal Peti Kemas

5.3 Hasil Implementasi Pembentukan Business Process Model

Di bab ini akan dijelaskan mengenai hasil implementasi Pembentukan Business Process Model. Hasil ini akan dibagi 3 yaitu hasil implementasi pembentukan business process model dari *Declarative Model*, Neo4J, dan *Heuristic Miner*.

5.3.1 Hasil Implementasi Pembentukan Business Process Model dari Model Deklaratif

Didapatkan hasil implementsi dari metode yang dijalankan . Terdapat 5 jenis LTL disini yaitu LTLFirstLast untuk mendapatkan aktivitas pertama dan terakhir, LTL Sequence untuk mendapatkan aktivitas yang berurutan, LTL XORSplit untuk mendapatkan aktivitas yang mempunyai relasi xor split, LTL XorJOIN untuk mendapatkan aktivitas yang mempunyai relasi xor join, LTL NonFreeChoice untuk mendapatkan aktivitas yang mempunyai relasi non free choice. Isi dari setiap jenis LTL bisa dilihat dari Tabel 5.1 .

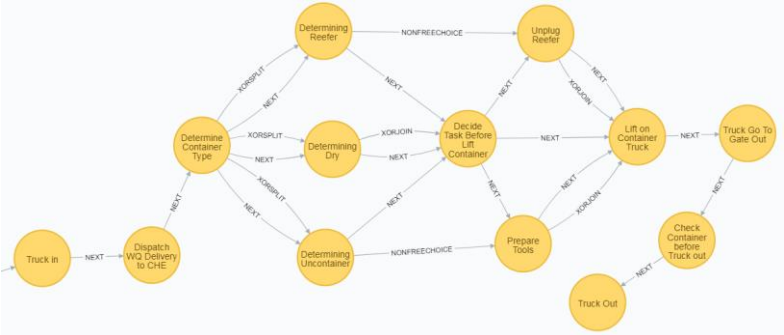
Tabel 5.1. Hasil LTL

Jenis LTL	Isi LTL
LTLFirstLast	Firstactivity(Entry Document via PDE-complete) Lastactivity(Truck Out-complete)
LTLSequence	Discharge Yard Planning-complete -> O(Assign Working Instruction to CC and Yard-complete) Entry Document via PDE-complete -> O(Vessel Berthing Process-complete) Bring Container to Behandle Area-complete -> O(Check Goods Behandle-complete) Create Job Order Document Delivery-complete -> O(Truck in-complete) Truck Go To Gate Out-complete -> O(Check Container before Truck out-complete) Receive Container-complete -> O(Stack Container in Yard-complete) Assign Working Instruction to CC and Yard-complete -> O(Discharge Container-complete) Lift on Container Truck-complete -> O(Truck Go To Gate Out-complete) Check Goods Behandle-complete -> O(Bring Back To Yard from Behandle-complete) Bring Back To Yard from Behandle-complete -> O(Create SPPB-complete) Vessel Berthing Process-complete -> O(Discharge Yard Planning-complete) Create Job Order Document Behandle-complete -> O(Bring Container to Behandle Area-complete)

Jenis LTL	Isi LTL
	Truck in-complete -> O(Dispatch WQ Delivery to CHE-complete) Verification Document-complete -> O(Create document SPPB-complete) Discharge Container-complete -> O(Receive Container-complete) Create Job Order Document Quarantine-complete -> O(Bring Container to Quarantine Area-complete) Check Goods Quarantine-complete -> O(Bring Back To Yard from Quarantine-complete) Bring Container to Quarantine Area-complete -> O(Check Goods Quarantine-complete) Check Container before Truck out-complete -> O(Truck Out-complete) Dispatch WQ Delivery to CHE-complete -> O(Determine Container Type-complete)
LTLXORSplit	Determine Category-complete -> O(Create Job Order Document Behandle-complete \/ Verification Document-complete) Stack Container in Yard-complete -> O(Determine Category-complete \/ Create Job Order Document Quarantine-complete) Determine Container Type-complete -> O(Determining Reefer-complete \/ Determining Dry-complete \/ Determining Uncontainer-complete)
LTLXORJoin	O(Bring Back To Yard from Quarantine-complete \/ Stack Container in Yard-complete) -> O(Determine Category-complete)

Jenis LTL	Isi LTL
	O(Create SPPB-complete \/ Create document SPPB-complete) -> O(Create Job Order Document Delivery-complete) O(Prepare Tools-complete \/ Decide Task Before Lift Container-complete \/ Unplug Reefer-complete) -> O(Lift on Container Truck-complete)
LTLNonFreeChoice	<>((Lift on Container Truck-complete /\ Decide Task Before Lift Container-complete /\ Determining Dry-complete) \/ (Determining Uncontainer-complete /\ Decide Task Before Lift Container-complete /\ Prepare Tools-complete) \/ (Determining Reefer-complete /\ Decide Task Before Lift Container-complete /\ Unplug Reefer-complete))

Setelah mendapatkan LTL itu kemudian diubah ke dalam cypher dan hasilnya dapat di lihat di Gambar 5.4.



Gambar 5.4. Hasil Graph dari LTL

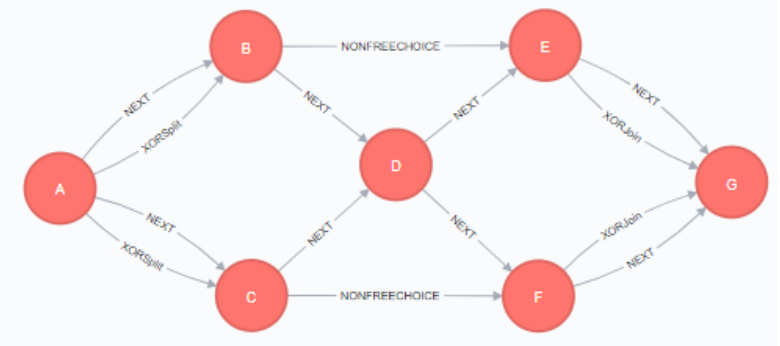
Kasus lain yaitu ada trace yang berisi [ABDEG, ACDFG ABDEG]. Hasil LTL yang didapatkan dapat dilihat pada table 5.1.

Tabel 5.2. Hasil LTL dari *Declarative Miner*

Table	LTL List

LTLFirstLast	*Firstactivity*(A)
	Lastactivity(G)
LTLXORSplit	A -> O((B ∨ C))
LTLXORJoin	O((E ∨ F)) -> O(G)
LTLNonFreeChoice	<>((B ∧ D ∧ E) ∨ (C ∧ D ∧ F))

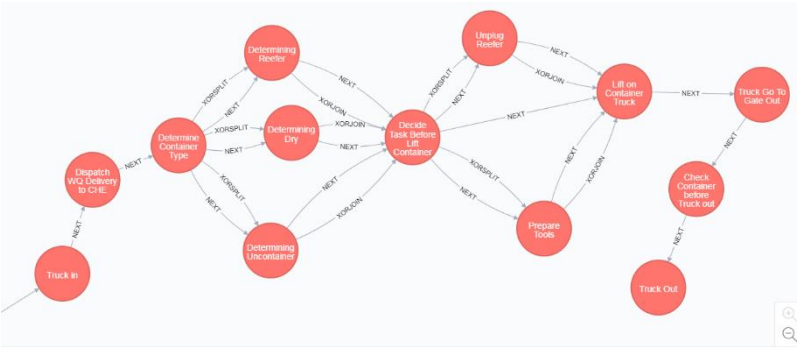
Setelah mendapatkan hasil LTL di atas, lalu bentuk LTL tersebut ke dalam *graph model* dan didapatkan hasil pada gambar 5.5.



Gambar 5.5. Hasil *graph model* dari LTL

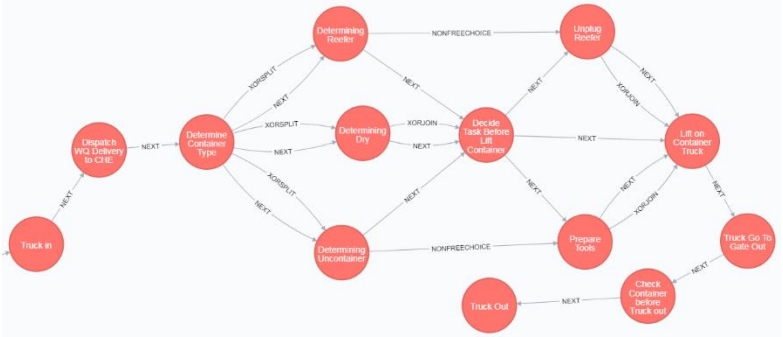
5.3.2 Hasil Implementasi Pembentukan Business Process Model dari Model Neo4J

Gambar 5.6 menunjukkan Business Process Model dalam bentuk graph.



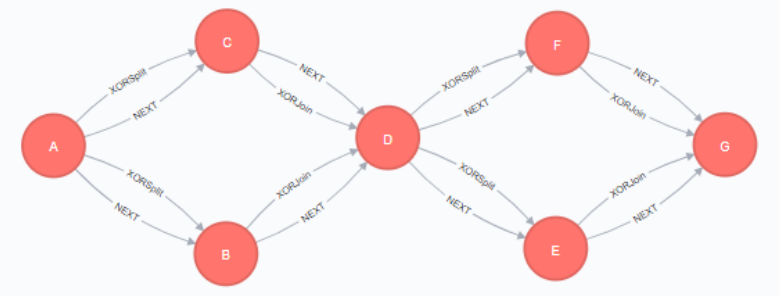
Gambar 5.6. Hasil *Graph model* dari *Event Log*

Gambar 5.7 menunjukkan Business Process Model setelah didapatkan relasi *non-free choice.*



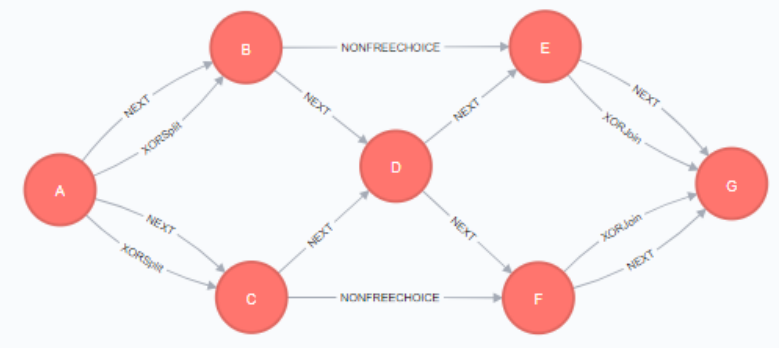
Gambar 5.7. Hasil *Graph model* dari *Event Log* setelah penambahan relasi *non-free choice*

Kasus lain yaitu ada trace yang berisi [ABDEG, ACDFG ABDEG]. Dari trace tersebut dibuat *graph model* yang dapat dilihat pada gambar 5.8.



Gambar 5.8. Hasil *graph* dari *Event Log*

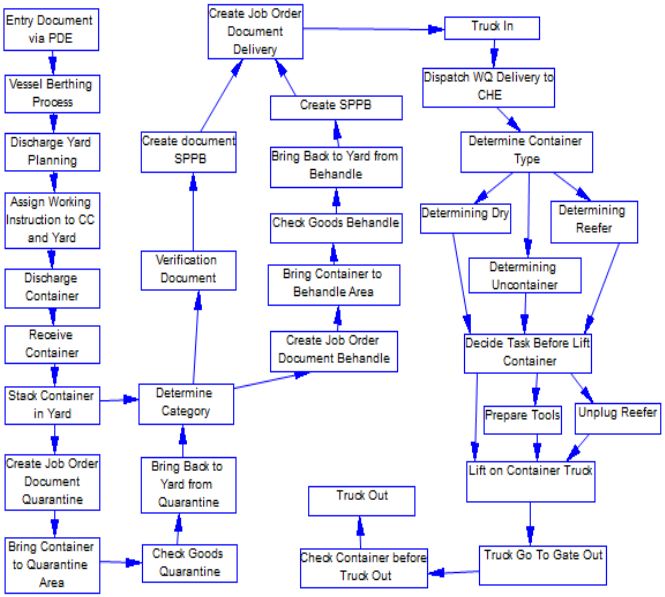
Setelah itu kita mencari relasi *non-free choice* dan didapatkan hasil pada gambar 5.9



Gambar 5.9. Hasil *Graph model* dari *Event log* setelah penambahan relasi *non-free choice*

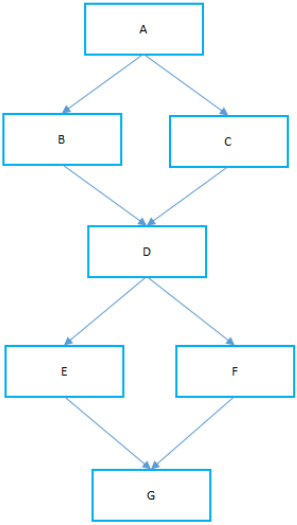
5.3.3 Hasil Implementasi Pembentukan *Business Process Model* dari *Model Heuristic Miner*

Setelah melakukan metode untuk melakukan pembentukan Business Process Model dari Model *Heuristic Miner*, maka akan keluar hasil seperti pada Gambar 5.10.



Gambar 5.10. Hasil *Graph* dari *Heuristic Miner*

Kasus lain yaitu ada trace yang berisi [ABDEG, ACDFG ABDEG]. Dari trace tersebut dibuat *graph model* yang dapat dilihat pada gambar 5.11.



Gambar 5.11. Hasil *Graph* dari *Heuristic Miner*

5.4 Hasil Perbandingan 3 Graph

Dari pengolahan LTL dari *Declarative Miner*, *event log* langsung dan *Heuristic Miner* didapatkan hasil terbaiknya adalah hasil Neo4j dan hasil LTL. Hal ini karena *Heuristic Miner* tidak bisa langsung menghubungkan relasi antar aktivitas. Sedangkan untuk mendapatkan relasi kita harus konversi terlebih dahulu ke petri net. Meski kita mendapatkan relasi antar aktivitas, relasinya masih salah jika event log mengandung *non-free-choice*. Misalnya hubungan antara B ke E dan C ke F pada Gambar 5.11. Hubungan antara B ke E dan C ke F harusnya memiliki relasi *non-free choice*. Sebaliknya, hasil Neo4J bisa langsung menghasilkan relasi yang benar padahal *event log* mengandung *non-free choice*.

BAB VI
KESIMPULAN DAN SARAN

Pada bab ini akan diberikan kesimpulan yang diambil selama pengerjaan Tugas Akhir serta saran tentang pengembangan yang dapat dilakukan terhadap Tugas Akhir ini di masa yang akan datang.

6.1 Kesimpulan

Dari hasil pengamatan selama proses perancangan, implementasi, dan pengujian perangkat lunak yang dilakukan, dapat diambil kesimpulan sebagai berikut:

1. Data dari *database* ditransform dengan baik dan benar menjadi *event log* yang dapat diolah.
2. Pembentukan *rule LTL* terhadap log data yang mengandung *non-free choice* dapat dilakukan dengan menggunakan *Declarative Miner*
3. Pembentukan *graph model* dari *LTL* dapat dilakukan dengan menggunakan Neo4J.
4. *Graph Model* yang dibentuk dari *event log* dengan menggunakan *Neo4J* dapat mengatasi *non-free choice.*
5. *Graph Model* yang dibentuk dari *event log* dengan menggunakan *Heuristic Miner* tidak dapat mengatasi *non-free choice.*
6. Metode menggunakan *Neo4J* dan *Declarative Miner* merupakan metode yang paling tepat untuk menggambarkan *Business Process Model.*

6.2 Saran

Berikut merupakan beberapa saran untuk pengembangan sistem di masa yang akan datang. Saran ini didasarkan pada hasil perancangan, implementasi dan pengujian yang telah dilakukan. Saran tersebut adalah perbaikan pada algoritma *Heuristic Miner* agar dapat menangani *event log* yang mengandung *Non-free Choice.*

[Halaman ini sengaja dikosongkan]

DAFTAR PUSTAKA

[1] L. Wen, J. Wang, W. M. P. van der Aalst, B. Huang, and J. Sun, "Mining process models with prime invisible tasks," Data & Knowledge Engineering, vol. 69, no. 10, pp. 999–1021, Oct. 2010. https://doi.org/10.1016%2Fj.datak.2010.06.001

[2] R. Sarno, R. D. Dewandono, T. Ahmad, M. F. Naufal, and F. Sinaga, "Hybrid Association Rule Learning and Process mining for Fraud Detection," *IAENG Int. J. Comput. Sci.*, vol. 42, no. 2, pp. 59–72, 2015.

[3] R. Sarno, Y. A. Effendi, and F. Haryadita, "Modified Time-Based Heuristics Miner for Parallel Business Processes," *Int. Rev. Comput. Softw.*, vol. 11, no. 3, pp. 249–260, 2016. https://doi.org/10.15866/irecos.v11i3.8717

[4] N. Y. Setiawan and R. Sarno, "Multi-Criteria Decision Making for Selecting Semantic Web Service Considering Variability and Complexity Trade-Off," *J. Theor. Appl. Inf. Technol.*, vol. 86, no. 2, pp. 316–326, 2016.

[5] S. R.a *et al.*, "Developing a workflow management system for enterprise resource planning," *IAENG Int. J. Comput. Sci.*, vol. 72, no. 3, pp. 412–421, 2015.

[6] R. Sarno and K. R. Sungkono, "Coupled Hidden Markov Model for Process Mining of Invisible Prime Tasks," *Int. Rev. Comput. Softw.*, vol. 11, no. 6, pp. 539–547, 2016. https://doi.org/10.15866/irecos.v11i6.9555

[7] R. Sarno and K. R. Sungkono, "Hidden Markov Model for Process Mining of Parallel Business Processes," *Int. Rev. Comput. Softw.*, vol. 11, no. 4, pp. 290–300, 2016. https://doi.org/10.15866/irecos.v11i4.8700

[8] S. Huda, R. Sarno, and T. Ahmad, "Increasing accuracy of process-based fraud detection using a behavior model," *Int. J. Softw. Eng. its Appl.*, vol. 10, no. 5, pp. 175–188, 2016. https://doi.org/10.14257/ijseia.2016.10.5.16

[9] W. Chomyat and W. Premchaiswadi, “Process mining on medical treatment history using conformance checking,” 2016. https://doi.org/10.1109/ictke.2016.7804102

[10] T. Erdogan and A. Tarhan, “Process Mining for Healthcare Process Analytics,” 2016. https://doi.org/10.1109/iwsm-mensura.2016.027

[11] A. S. Osses, “Business process analysis in advertising: An extension to a methodology based on process mining projects,” 2016. https://doi.org/10.1109/sccc.2016.7836000

LAMPIRAN

[Halaman ini sengaja dikosongkan]

DAFTAR ISTILAH

AND	:	relasi yang menunjukkan bahwa seluruh aktivitas pada relasi ini harus dijalankan semua di tiap proses.
Case	:	proses
Invisible Non-Prime Task	:	aktivitas bukan utama tak terlihat
Non-free choice	:	pilihan tidak bebas
OR	:	relasi yang menunjukkan bahwa seluruh aktivitas pada relasi tersebut dapat dijalankan semua atau dipilih beberapa untuk tiap proses
Node	:	unit dalam graf
Precision	:	nilai presisi
Process Discovery	:	pembentukan model proses
Process Mining	:	penggalian proses
Skip activity	:	aktivitas yang hilang
SOP	:	*Standard Operating Procedure*
Trace	:	varian proses pada log data
Truncated Process	:	proses yang terpotong
XOR	:	relasi yang menunjukkan bahwa seluruh aktivitas pada relasi ini harus dipilih salah satu untuk dijalankan
YAWL	:	*Yet Another Work flow Language*

[Halaman ini sengaja dikosongkan]

BIODATA PENULIS

Andy Yohanes, lahir pada tanggal 10 Juli 1996 di Lumajang. Penulis merupakan seorang mahasiswa yang sedang menempuh studi di Jurusan Teknik Informatika Institut Teknologi Sepuluh Nopember (ITS). Memiliki beberapa hobi antara lain bermain musik dan mendengarkan musik. Pernah menjadi pengurus PMK di ITS. Selama menempuh pendidikan di kampus, penulis juga aktif dalam organisasi kemahasiswaan, yaitu Staff Teknis National Logic Competition pada tahun ke-3.

[Halaman ini sengaja dikosongkan]